GenRad

TECHNICIAN FILE COPY

Test Library Programming

36535

227X Board Test Workstations

Model: 227

(Release 7.4)

Form 2272-0147-D

Instruction Manual

Contents

Introduction – Section 1
Automatic Test Generation Process – Section 2
Library Management User's Guide – Section 3
Test Librarian – Section 4
Analog Test Library Programming Guide – Section 5
Digital Test Library Programming Guide – Section 6
Analog Components Library User's Guide – Section 7
Library Routines – Section 8
Appendix
Index

TECHNICIAN FILE COPY

Test Library Programming

227X Board Test Workstations

36535
model
2275

(Release 7.4)

Form 2272-0147-D

©GenRad 1983
Concord, Massachusetts, U.S.A. 01742
November, 1983

GenRad

List of Effective Pages

INSERT LATEST CHANGED PAGES. DESTROY SUPERSEDED PAGES.

NOTE: The portion of the text affected by the changes is indicated by a vertical line in the outer margin of the page.

TOTAL NUMBER OF PAGES IN THIS PUBLICATION IS 236 CONSISTING OF THE FOLLOWING:

Issue Original Date November, 1983

Page No.	Issue
Title	November, 1983
A	November, 1983
iii through vi	Original
1-1 through 1-2	Original
2-1 through 2-9	Original
3-1 through 3-8	Original
4-1 through 4-11	Original
5-1 through 5-48	Original
6-1 through 6-99	Original
7-1 through 7-3	Original
8-1 through 8-30	Original
A-1 through A-11	Original
I-1 through I-3	Original

NOTE

The software package, described in this manual and supplied by GenRad, Inc. as a component of a system or as a separate item, is furnished for use on a single CPU. All proprietary rights are reserved by GenRad, Inc.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Computer Software clause in DAR 7-104.9(a).

GenRad Inc., Concord, MA

The following are trademarks of GenRad, Inc.:

BUSBUST	SCRATCHPROBING
GRnet	SpeedPlus
NIM	TRACS

NOTE

The material in this manual is for informational purposes only and is subject to change without notice. GenRad assumes no responsibility for any errors or for consequential damages that may result from the use or misinterpretation of any of the procedures in this publication.

*The asterisk indicates pages changed, added, or deleted by the current change.

Table of Contents

Section and Paragraph		Page
1 INTRODUCTION		
1.1	General	1-1
2 AUTOMATIC TEST GENERATION (ATG)		
2.1	General	2-1
2.2	Circuit Description Compiler (CDL)	2-1
2.3	Circuit Description Extractor (CDE)	2-4
2.4	Analog Auto Test Generator (AATG)	2-4
2.5	Digital Test Generator (DTG)	2-9
2.6	Merge	2-9
3 LIBRARY MANAGEMENT USER'S GUIDE		
3.1	General	3-1
3.2	Developing a New User Library	3-1
3.3	Modifying Existing Library Files	3-6
3.4	Listing a Library File	3-7
4 TEST LIBRARIAN		
4.1	General	4-1
4.2	Notes on the Monitor	4-1
4.3	Analog Test Librarian (ATL)	4-1
4.4	Digital Test Librarian (DTG)	4-5
4.5	Analog Component Librarian (ACL)	4-8
5 ANALOG TEST LIBRARY PROGRAMMING		
5.1	Introduction	5-1
5.2	Analog Library Organization Statements	5-1
5.3	Analog Test Library Processing	5-9
5.4	Developing an Analog Test Procedure	5-13
5.5	Analog Test Selection Language (ATSL)	5-24
5.6	Test Format Instructions	5-41
5.7	Considerations in Library Design	5-45
6 DIGITAL TEST LIBRARY		
6.1	Introduction	6-1
6.2	Writing a Digital Library Procedure	6-6
6.3	Digital Test Structure Language	6-51
6.4	Sample Test Procedure	6-95
7 ANALOG COMPONENT LIBRARY USER'S GUIDE		
7.1	General	7-1
7.2	Writing a Component Description	7-2
8 SYSTEM LIBRARY TESTS		
8.1	General	8-1
8.2	ATG Tests	8-2
APPENDIX A — MICROPROCESSOR TESTS		
A.1	Introduction	A-1
A.2	Testing the 80/85 Processor Family	A-1
A.3	Testing the 8048 Microprocessors	A-4
A.4	Testing the 8086 Microprocessors	A-5
A.5	Testing 6500 Microprocessor Family	A-9
A.6	Testing the TMS9900	A-9
INDEX		

Digitized by the Internet Archive
in 2024

https://archive.org/details/manualsplus_02478

Foreword

PURPOSE OF MANUAL

This manual is directed to those manufacturing test personnel who are advanced test programmers. It describes the techniques and the languages used for writing test procedures and inserting them into a user library. The sections of the manual present library information as follows:

Section 1 Introduction introduces the types of library files used in the test generation process.

Section 2 Automatic Test Generation describes the process of automatic test generation and how the library files are used.

Section 3 Library Management User's Guide describes the process of creating and modifying library files.

Section 4 Test Librarian defines the monitor page options used in library management.

Section 5 Analog Test Library Programming Guide describes the language used to write test library routines for analog components.

Section 6 Digital Test Library Programming Guide describes the language used to write test library routines for digital components.

Section 7 Analog Components Library Programming Guide describes how to represent complex analog devices in a components library.

Section 8 System Library Tests describes test routines of device types that exist in the system libraries.

Appendix A Microprocessor Tests describes techniques for testing microprocessors where the routines require user-added hardware.

RELATED DOCUMENTS

Other manuals that may be needed for information on creating and running test software are:

Test Programming manual
Test Set Preparation manual
System Operation manual for the appropriate system model

Refer to the Document Directory for a complete list of manuals in the document set. The Document Directory is useful in finding information within the set of manuals.

DEFINITION OF TERMS

Systems -- A term that refers to the workstation and related software and options.

Workstation -- The words workstation and system may be used interchangeably in this manual.

DIFFERENCE DATA

Release 7.4 has several new features that improve the user interface to the system and that provide expanded testing capability. The primary new feature that is of concern to library programming is the SpeedPlus enhancement.

SpeedPlus

2271 and 2272 workstations may be purchased or retrofitted with the SpeedPlus option. This option provides for a programmable clock signal of up to 20 MHz and driver/sensor test step rates of up to 5 MHz. Programmable clock synchronization is also provided for synchronizing to on-board clocks. Triggering is also provided for keying drive/sense timing to specific events. The clock and trigger statements are described in the 227X Test Programming manual. Synchronizing to on-board clocks is generally accomplished via commands in the .ATO file. Refer to Section 6 for instructions on writing SpeedPlus library routines. Devices for which SpeedPlus routines have been added to the digital library by GenRad are indicated in Appendix A of the Test Set Preparation manual.

Introduction – Section 1

1.1 GENERAL

The Automatic Test Generation (ATG) feature of the GenRad 227X Board Test Workstation develops test programs from the repertoire of test descriptions in the test libraries. There are 3 types of libraries included in the system software:

1. Analog Test Library (ATL) containing test procedures for basic characteristics of analog circuit devices
2. Digital Test Library (DTL) containing test procedures for a large quantity of digital IC components
3. Analog Component Library (ACL) containing analog descriptions of special and complex components.

The first phase of the ATG process creates a circuit data file. This file automatically interacts with the test library routines to develop test procedures for each component type. Most tests will be drawn from the 3 supplied system libraries; for unusual component types, however, users can add user-created sections to each of the libraries.

The Analog Test Library contains procedures for testing circuit continuity and basic electrical parameters such as resistance, capacitance, inductance, transistor gain, etc. The ATG process searches the Analog Test Library for a test for each analog component mentioned in the circuit data file. For special analog devices, users may devise their own tests and place them in a user section of the ATL.

The Digital Test Library contains test procedures for most commonly used digital IC types. For other IC types, users can modify procedures in the system library or develop entirely new procedures and place them in a user DTL. For each digital IC device found in a circuit data file, ATG attempts to find a corresponding procedure in the DTL.

The Analog Component Library contains descriptions of components that ATG does not normally recognize. The ACL description reduces a component to one or more equivalent basic components that can be recognized by the ATG software. A few standard complex components are described in the system Analog Component Library; others may be added to a user ACL. During the first phase of ATG the ACL is searched for non-standard components, which are then expanded into their primitive components for further ATG processing.

Systems are provided with a system library file of each of the 3 types. These libraries cannot be modified by the user. It is expected, however, that users will want to customize and update their libraries. Therefore, the system provides a capability for creating Update and User library files for each of the 3 types. The primary purposes for creating additional library files are to:

Add new test procedures provided by GenRad software releases
Add user-created test procedures
Modify tests that exist in the system library files

Automatic Test Generation Process - Section 2

2.1 GENERAL

Automatic Test Generation (ATG) consists of several tasks. The purposes of these tasks are as follows:

Circuit Description Compiler (CDL)	Compile the circuit description source (.CKT) file into a circuit description work (.WOR) file, and include applicable descriptions from .ACL.
Circuit Description Extractor (CDE)	Create a .DAI file which contains nail data, bused node data, and undrivable IC data. Create an .IDD or .IDX file which contains component pin data, node lists and driver/sensor data. The .IDD is also used for run-time repair messages.
Analog Auto Test Generator (AATG)	Work with the Analog Test Library (.LIB) files, the Automatic Test Options (.ATO or .ATX) files, and the (.WOR) file to analyze circuits and to generate an Analog Test Program (.ATP).
Digital Test Generator (DTG)	Work with the Digital Test Library (.DTL) files, the .DAI file and the .IDD or .IDX file to structure tests for the digital components and to generate a Digital Test Program (.DTP).
Merge	Merge the .DTP into the .ATP to result in an overall test program (.TPG or .TPX).

The relationship of these tasks and the various files and reports generated are shown in Figure 2-1.

Basic descriptions of the 5 tasks are provided in the remaining paragraphs in this section.

2.2 CIRCUIT DESCRIPTION COMPILER (CDL)

The first system software task that must be performed in the process of automatically generating a test program is the circuit description compiler (CDL) which compiles the circuit description source file (.CKT). The compiled file is called the circuit description work file (.WOR). This .WOR file sets up the circuit description data in a format that can be read by the analog auto test generator (AATG) and the circuit description extractor (CDE).

The CDL processes each item as it appears in the .CKT file. The first item in any .CKT file is the %CIRCUIT; statement. This will set up a table header under which all components will be listed. Table sub-headers will be set up for each component type the first time that type is encountered. All components of each type will be listed under the appropriate sub-heading. If a complex component is encountered, a sub-heading is made for that component under the CIRCUIT header, and a new top level header is set up so that the complex component can be broken down into its component types with appropriate sub-headers.

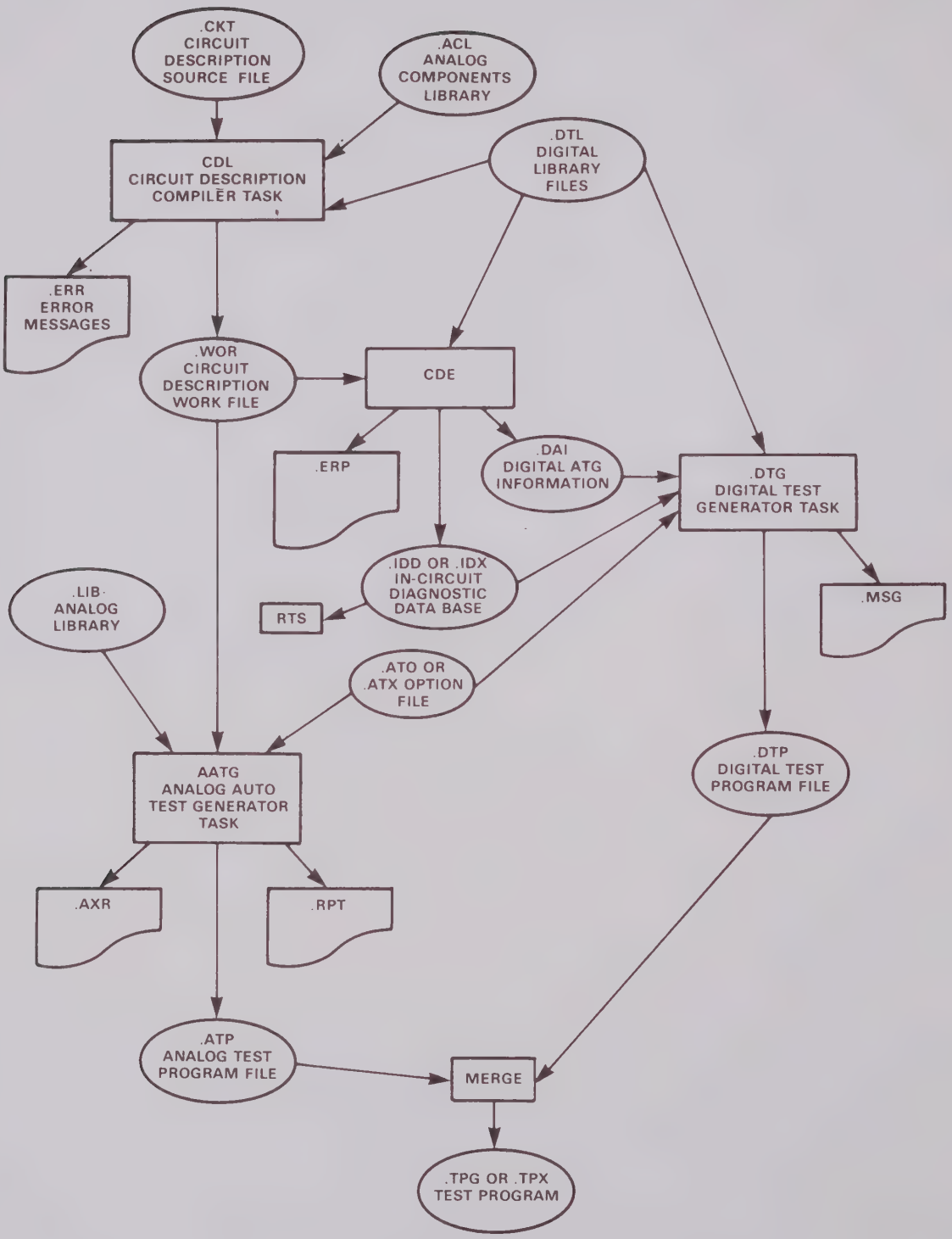


Figure 2-1. ATG-Process Flow Diagram.

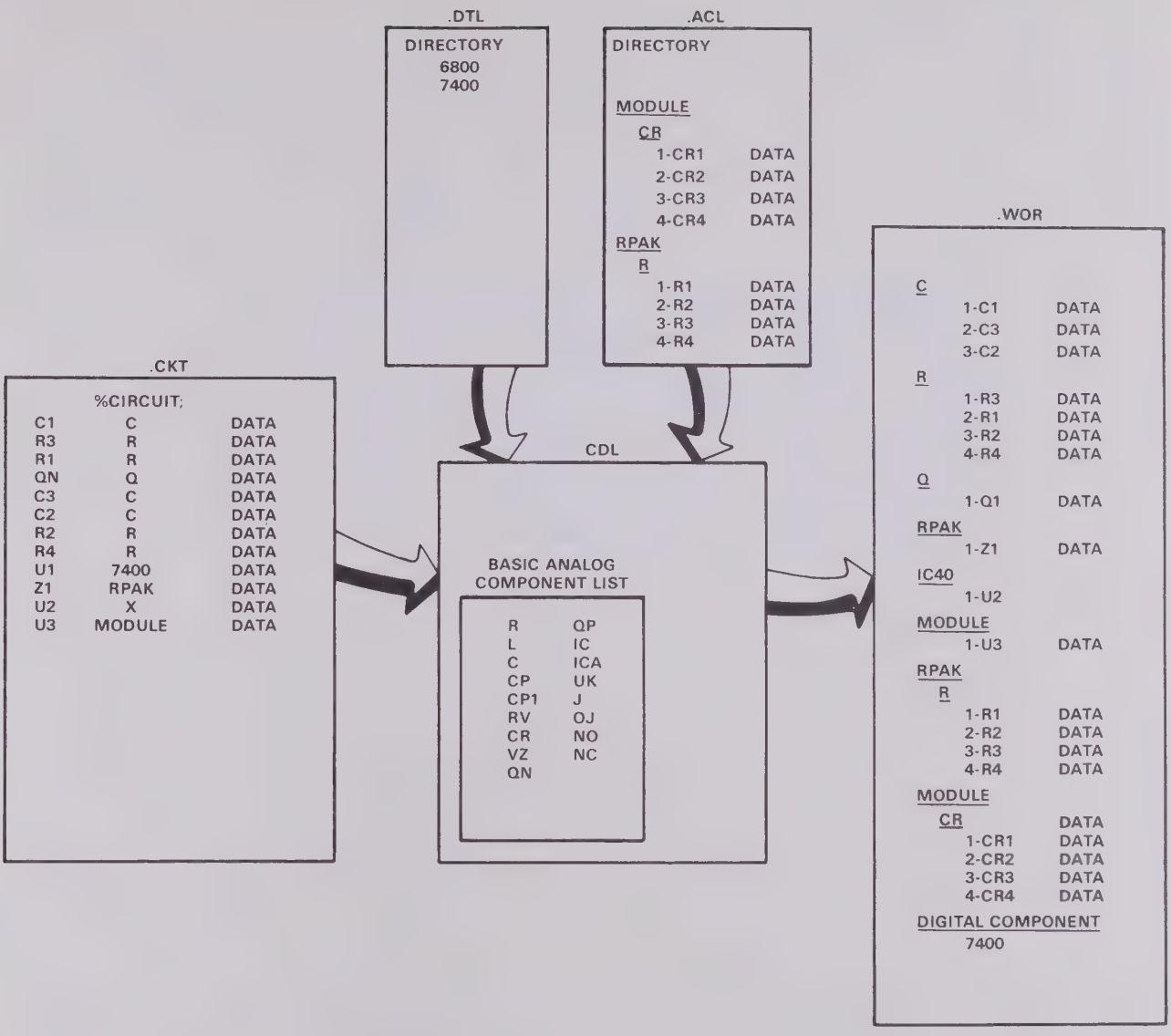


Figure 2-2. CDL Processing.

The CDL task looks at each component listed in the CIRCUIT section of the .CKT file. It then identifies the type of each component listed. CDL has a list of basic analog component types stored in its memory. This list is checked first. If the component type is found, the .CKT data for that component are transferred to the .WOR file under the sub-heading for that type of component. If the component type is not found in CDL's internal list, CDL looks in the digital library. If found there, the name of the device is entered in the .WOR file under the major heading of digital component. Whether or not the component type was found in the digital library, CDL also looks at the Analog Component Library. If found there, the name of the component is entered in the .WOR file and, if it is the first occurrence of the component, the name becomes a new major heading. If the component type is not found in any library, it is considered to be a 40-pin IC and is entered under the sub-heading IC40. Figure 2-2 shows an example of CDL processing and the order of components contained in the .WOR file.

2.3 CIRCUIT DESCRIPTION EXTRACTOR (CDE)

The circuit description extractor task performs 2 functions. One function is to compile a list of data from the .WOR and the Digital Test Library (.DTL) that is put into an In-Circuit Diagnostic Data (.IDD or .IDX) file used for run-time diagnostics. These files are also used by the Digital Test Generator (DTG). The other function preprocesses digital circuit information for DTG and lists it in the Digital ATG Information (.DAI) file. Data in these files are as follows:

.IDD File
: Component pin to driver/sensor and analog scanner interconnections. Pin usage, such as analog, digital input, or digital output.

.IDX File
: Same as .IDD except that nail data are temporary. The nail assign task will assign actual nails for 2271 and 2272 systems and create an .IDD file.

.DAI File
: Driver/sensors
: Bused nodes
: Undrivable ICs

2.4 ANALOG AUTO TEST GENERATOR (AATG)

2.4.1 Introduction

The Analog Test Library contains test procedures which the Analog ATG Task uses in developing test programs for a circuit board. There is a library test procedure for each type of component characteristic that is to be considered for Automatic Test Generation. These test procedures consist of a basic test format plus decision-making algorithms which cause a measurement to be chosen. These measurement configurations take into consideration all the equivalent circuit characteristics of circuit elements in the vicinity of the component being tested.

There may be component types on a circuit board which are not considered in the library. There may also be circuit configurations where ATG will decide that damage to components in the circuit could occur and therefore will not write a test. ATG will also not write a test if it decides that the circuit configuration is such that there would be excessive error in the measurement. In such cases, the user has a choice of modifying the program that ATG produced, or of making changes to the test library. The following paragraphs describe the AATG process and library organization. The language that is used for writing library selection routines is contained in Section 5.

2.4.2 Library Organization

The Analog Test Library consists of three files. One of these, the system analog test library file, is included in the original system software. This file may be copied but not modified by the user. The other two, Update and User Test Library files, may be created and modified by the user.

The Analog Test Library is organized with an outline and directories. The outline shows the existence of all available directories. A directory shows the existence of all available tests for the applicable component type or characteristic. Figure 2-3 shows this organization.

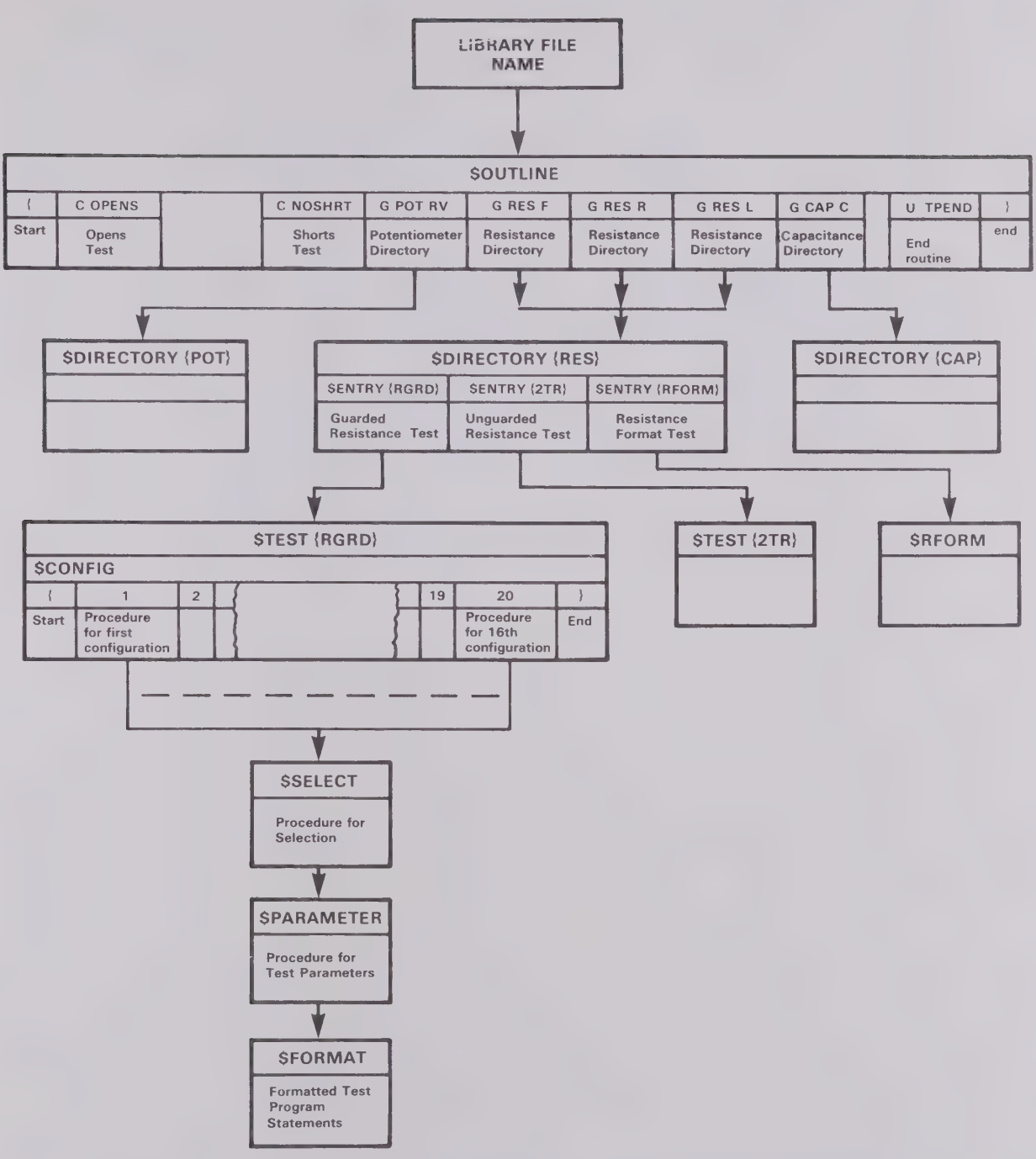


Figure 2-3. Analog Library Organization.

2.4.3 Analog ATG Processing

The AATG task takes one component at a time in the order in which it appears in the .WOR file and develops a test for that component under the guidance of the Analog Test Library. Guard sets are modeled and equivalent characteristics of the surrounding circuitry are calculated. The test procedure being considered uses the data from this circuit analysis to decide if this test procedure is acceptable. Prior to developing tests for the components listed in the .WOR file, AATG will write out connectivity tests for the whole board. After all the component tests are generated, all the utility procedures in the Analog Test Library are written.

A component that is described in an Analog Component Library produces 2 entries in the .WOR file. See Figure 2-2. The first of these entries falls under the main heading CIRCUIT and causes ATG to look in the Analog Test Library for a test of the overall component. The second entry is a main heading under which the individual basic components are listed. If a test exists in an Analog Test Library for the device as a whole entity, the AATG will develop its test at the time it encounters it. Once this test is generated, AATG will be directed to the name of the component listed as a main heading. ATG will then generate tests for each of the basic elements of the complex device as they are listed in the .WOR. Therefore, all tests for the component are done together so that statistics about the component are unified. A third level of component entries could result if a complex device is part of a larger complex device. AATG then returns to the next component in the .WOR file. Figure 2-4 shows a simplified Analog ATG Flow Diagram.

An explanation of Figure 2-4 provides an understanding of how ATG uses the library file organization described in Figure 2-3.

The first procedures to be written are the tests to check for continuity between circuit nodes. ATG also checks for discrepancies in the circuit description file (.CKT) and writes a file that presents circuit interconnection statistics and circuit discrepancies (.AXR). Processing may be stopped early if errors are found in continuity or in cross referencing. ATG looks for continuity tests that are in the Analog Test Library outline, such as the opens test (C OPENS) and the shorts tests (C NOSHRT), and writes them into the test program.

The component tests are processed by ATG looking in the .WOR file for a component. (The components are processed in the order in which they are listed in the .WOR.) ATG will then look for an OUTLINE. When one is found, it is searched for an appropriate DIRECTORY name. Then the libraries are searched for that DIRECTORY. When found, it looks for a TEST name. The TEST with that name is then searched for in the libraries. The test selection process, illustrated in Figure 2-5, is then performed for that TEST.

If a test procedure is selected, ATG goes to the $FORMAT section of the library $TEST to perform the routine which writes the test procedure into a temporary analog test program file (.ATZ).

Then, statistics about the test are written into the report file (.RPT). If a test is selected, the statistics will indicate a Superior or an Adequate rating. Otherwise, an Unsuccessful test or Not testable indication will be put in the report.

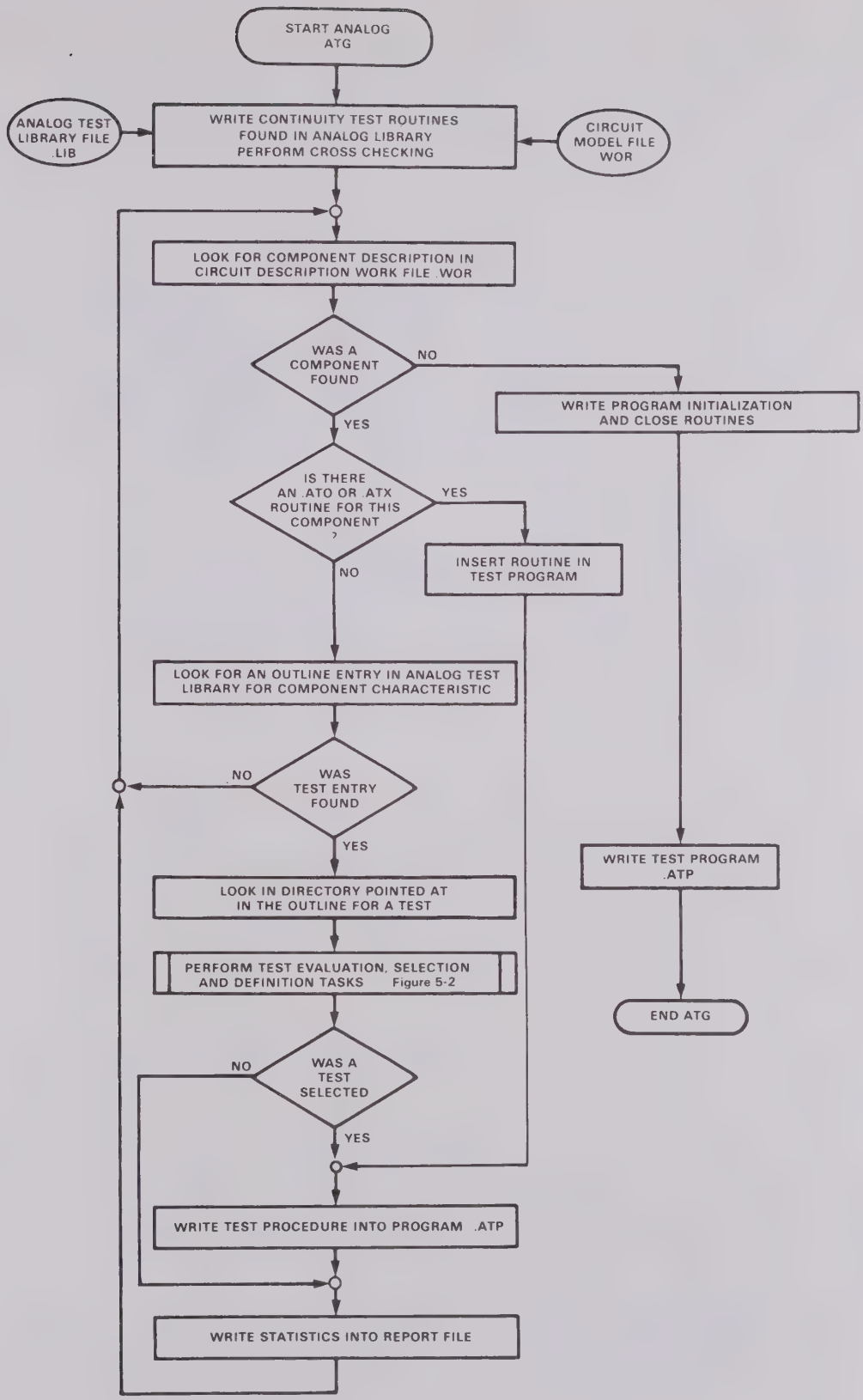

START ANALOG ATG
ANALOG TEST LIBRARY FILE .LIB
WRITE CONTINUITY TEST ROUTINES FOUND IN ANALOG LIBRARY PERFORM CROSS CHECKING
CIRCUIT MODEL FILE WOR
LOOK FOR COMPONENT DESCRIPTION IN CIRCUIT DESCRIPTION WORK FILE .WOR
WAS A COMPONENT FOUND
NO
YES
WRITE PROGRAM INITIALIZATION AND CLOSE ROUTINES
IS THERE AN .ATO OR .ATX ROUTINE FOR THIS COMPONENT ?
YES
NO
INSERT ROUTINE IN TEST PROGRAM
LOOK FOR AN OUTLINE ENTRY IN ANALOG TEST LIBRARY FOR COMPONENT CHARACTERISTIC
WAS TEST ENTRY FOUND
NO
YES
WRITE TEST PROGRAM .ATP
LOOK IN DIRECTORY POINTED AT IN THE OUTLINE FOR A TEST
PERFORM TEST EVALUATION, SELECTION AND DEFINITION TASKS Figure 5-2
END ATG
WAS A TEST SELECTED
NO
YES
WRITE TEST PROCEDURE INTO PROGRAM .ATP
WRITE STATISTICS INTO REPORT FILE

Figure 2-4. Simplified Analog ATG Flow Diagram.

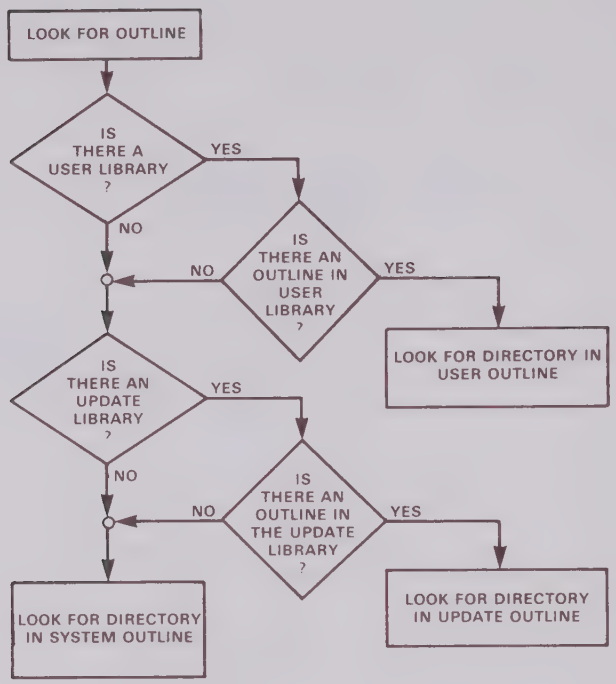


Figure 2-5. Analog Library Search Order.

When all the characteristics (applicable to the component type) in the $OUTLINE have been processed, the next component in the .WOR file will be processed. ATG will repeat the processing until all components of this type have been considered. Then ATG reorders the program in the temporary file and writes the collated result to a permanent Analog Test Program file (.ATP). The last ATG process is to combine this analog program with a digital test program, if it exists, and to generate a test program in a form recognizable by the run-time translator.

Now, if the system contains a User and/or an Update Library file, the processing control between libraries is modified, as explained in the following discussion.

When ATG is ready to start looking for test procedures in the various libraries, it will search for a library OUTLINE to control the ATG process for the complete board test. The OUTLINE that will be used will be that found in the highest-ordered library that contains an OUTLINE. Not all libraries must contain an OUTLINE, but an OUTLINE must exist in at least one library. If an OUTLINE does exist, it must contain all DIRECTORYs for all libraries because ATG will be controlled in its search for DIRECTORYs by the OUTLINE that it finds. The order of libraries, from highest order to lowest, is USER library, UPDATE library, and SYSTEM library. Figure 2-5 illustrates the search order in analog libraries.

As ATG steps through the controlling library OUTLINE, it will first look in the highest-ordered library for the indicated characteristic DIRECTORY. If not found, the search will jump to the next lower-ordered library. Once a characteristic DIRECTORY is found, processing control will remain with that

DIRECTORY until it is completed. Therefore, any DIRECTORY that exists must be complete for that characteristic. All ENTRYs in that DIRECTORY, however, do not require the corresponding TEST to be in that library; there may be tests with the same name in more than one library. The search for a TEST starts in the highest-ordered library; the first TEST found will be used.

2.5 DIGITAL TEST GENERATOR (DTG)

DTG creates a Digital Test Program (.DTP) that contains 2 types of test procedures. One type of procedure consists of tests for the individual ICs; the other consists of tests for each bus that exists on the circuit board. The bus test consists of 2 phases. The first determines if there is a fault that is causing the bus to be held in one state. The second phase is used if a fault does exist, to determine which IC output is causing it. In .DTP, bus testing precedes the testing of bused ICs so that the testing of any IC connected to a faulted bus can be bypassed at run time.

2.6 MERGE

The purpose of the Merge task is to generate an integrated Test Program (.TPG or .TPX) by combining the .ATP and the .DTP files. The .ATP contains the overall structure of the .TPG or .TPX and indicates where the .DTP is to be inserted. 2271 and 2272 systems normally generate a .TPX which contains temporary nail information that will be converted to actual test nail data by the nail assign task. The nail assign task creates the .TPG.

Library Management User's Guide – Section 3

3.1 GENERAL

There are separate library management programs, called librarians, for performing various library file management functions on the 3 types of library files. A library file is stored in a format that cannot be edited; therefore, individual library elements must translated into text files before they can be modified. All other operations on library files must be carried out via the librarian programs, which provide the following functions:

Add procedures to an Update or User library file from a source file.

Delete procedures from an Update or User library file.

Replace procedures in an Update or User library file with procedures from a source file.

Add, delete or replace an Outline or Directory in an Analog Update or User library file.

Add, delete or replace an alias in a Digital or Analog Component Update or User library file.

Copy procedures from any library to a corresponding Update or User library file.

Display procedures from any library on the video display.

Display outline or directories on the video display.

Display pin layouts of digital components.

Format a file for a new User or Update library.

List parts of any library file.

Procedures used in performing library management functions are described in paragraphs 3.2 through 3.4. Figures 3-1, 3-2, and 3-3 illustrate the relationship of library files and librarian functions. Librarian commands are fully defined in Section 4.

3.2 DEVELOPING A NEW USER LIBRARY

3.2.1 General

This section illustrates the steps necessary to make a test library file that contains procedures provided by the user. This user library file may better suit user needs than the system and update libraries. The 3 basic steps in creating a new user library are the same for each type of library.

1. Make a skeleton (empty, formatted file) for new library.
2. Provide source files for the new library.
3. Insert source files into the formatted file or copy existing items from a library to the new file.

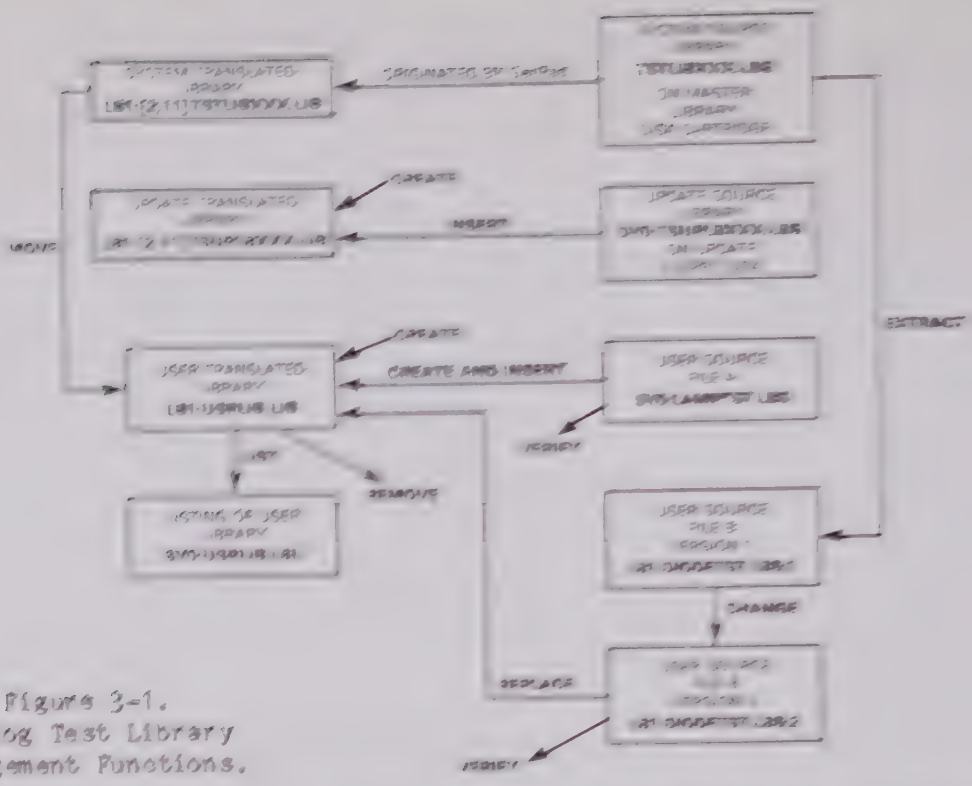


Figure 3-1.
Analog Test Library
Management Functions.

A movable segment of a library is called an item in the following discussions. A library item is different for each type of library. The items of an analog library are an outline, characteristic test directories, and test procedures. The items of a digital library are a directory, component test procedures, and component alias names. The items of an analog component library are a directory, component descriptions and description alias names.

Figure 3-2.
Digital Test Library
Management Functions.

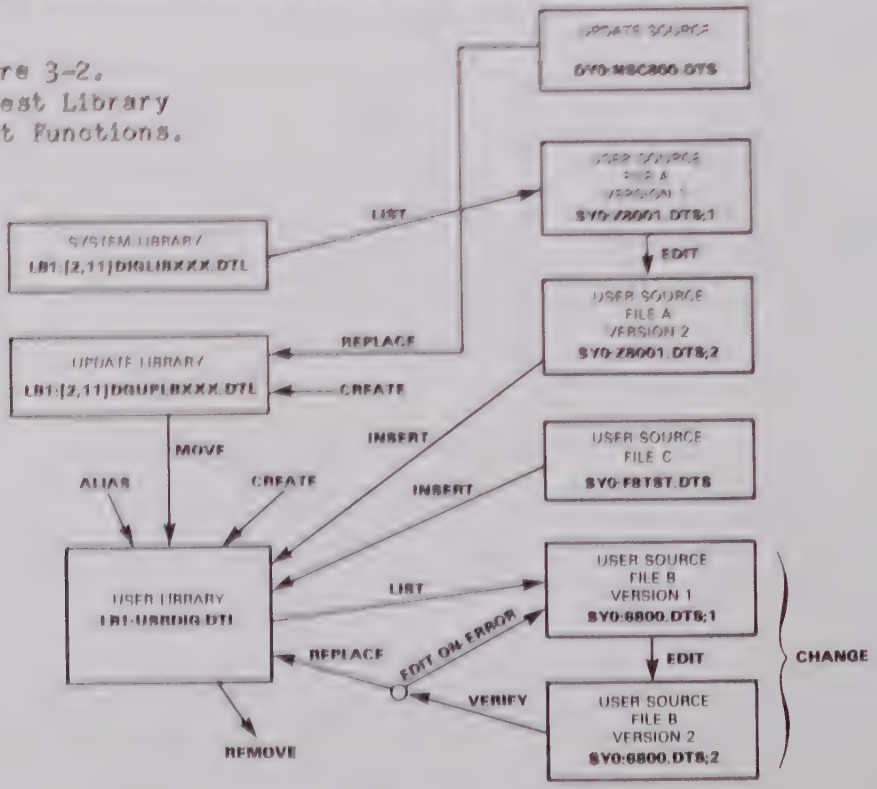
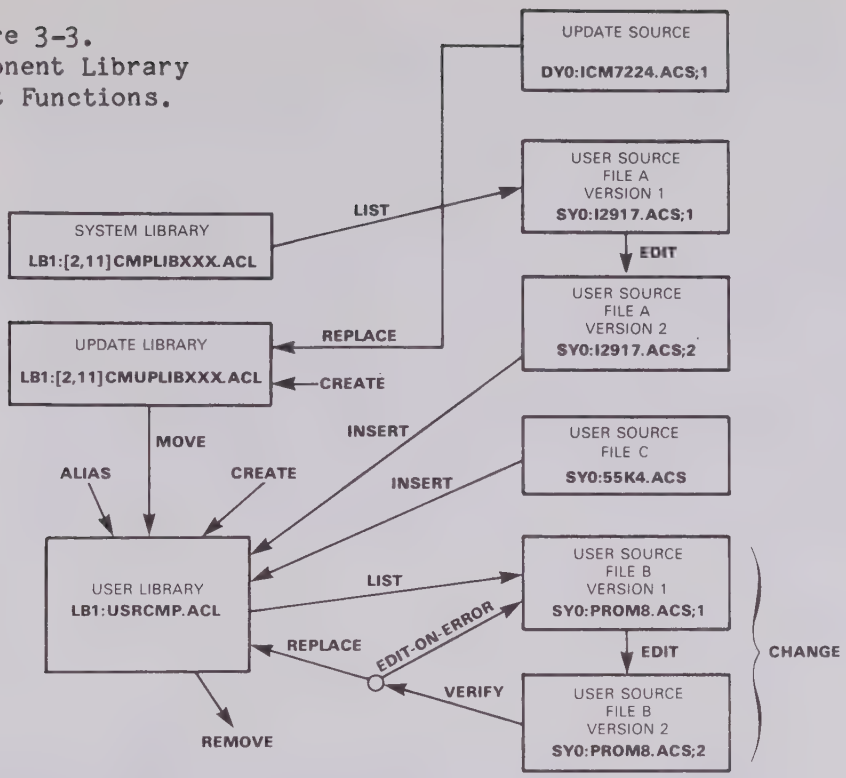


Figure 3-3.
Analog Component Library
Management Functions.



3.2.2 Create a Formatted File for Library Items

Before a library item can be put into a library file, the library file must be created. A library file is created by selecting CREATE mode on the appropriate librarian options page, specifying the name of the new file, and running the mode. This creates an empty formatted file which can accept an appropriate library item. Figure 3-4 illustrates typical options page specifications for the CREATE command.

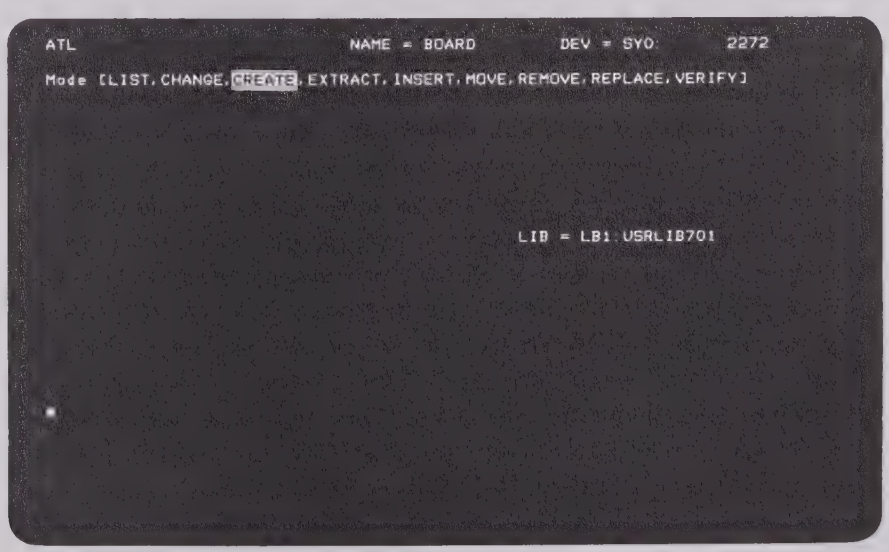


Figure 3-4. Example of ATL options page showing the command to create a formatted file for a new analog user library.

3.2.3 Prepare a Source File

Before a new item can be put into a library file, it must first exist in a source file. There are basically 2 methods of providing the source file:

1. Select the editor and type a new source file.

2. Copy an item from an existing library into a new source file and then select the editor for modifying that source file.

When developing a source file, it is important to remember that the Digital Test and Analog Component Librarians only work with one item at a time and, therefore, the source file may only contain one item. The Analog Test Librarian deals with any number of items and, therefore, the source file may contain any number of items.

The copying of an item from an existing library differs with each librarian. The Analog Test Librarian copies from a source file to a source file. A source file could not be made from a translated library. If the system analog library source file is to be copied, it has to be loaded into the system because it is normally stored off the system on a master disk. The ATL command to execute the copy function is EXTRACT. See Figure 3-5.

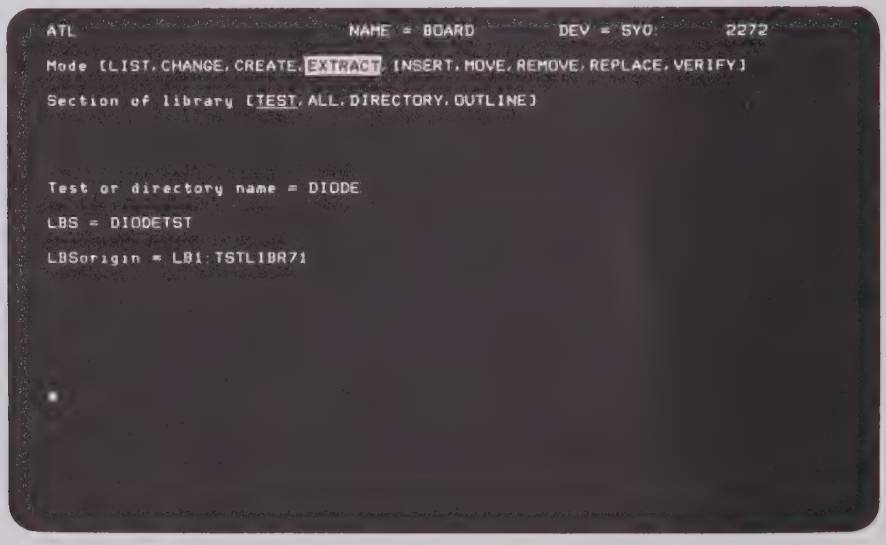

ATL NAME = BOARD DEV = SY0: 2272
Mode [LIST, CHANGE, CREATE, EXTRACT, INSERT, MOVE, REMOVE, REPLACE, VERIFY]
Section of library [TEST, ALL, DIRECTORY, OUTLINE]
Test or directory name = DIODE
LBS = DIODETST
LBSorigin = LB1:TSTLIBR71

Figure 3-5. Example of ATL Options page showing the command to copy a test from the system analog source library file to a new source file.

The Digital Test Librarian copies from a .DTL file because there is no translation process performed when making a library file from a source file. The DTL command to execute a copy is LIST. See Figure 3-6.

The Analog Component Librarian copies from an .ACL file because .ACL contains both the source listing and the compiled listing. The ACL command to execute a copy is LIST.

3.2.4 Insert Item(s) into the Library File

Once the source file is written, it is put into the new library file by selecting INSERT mode on the appropriate librarian options page, specifying the

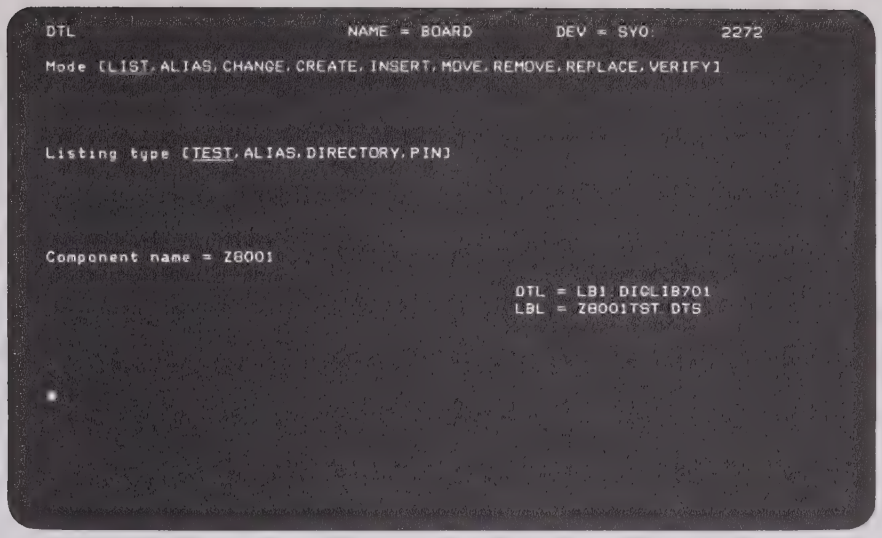

DTL NAME = BOARD DEV = SY0: 2272
Mode [LIST, ALIAS, CHANGE, CREATE, INSERT, MOVE, REMOVE, REPLACE, VERIFY]
Listing type [TEST, ALIAS, DIRECTORY, PIN]
Component name = Z8001
DTL = LB1:DIGLIB701
LBL = Z8001TST.DTS

Figure 3-6. Example of DTL options page showing the command to copy a test from the system digital test library file to a new source file.

item(s), and running the mode. Figure 3-7 illustrates typical options page specifications for the INSERT commands. The INSERT task varies with each library:

Analog

The INSERT task for analog procedures performs a syntax check of all the $CONFIG, $SELECT and $PARAMETER sections of source procedures, translates these procedures and provides translation error messages. If no errors occur, the translated procedure is inserted into the library file.

Digital

The INSERT task for digital procedures performs a syntax check of the source procedure. If there are no syntax errors, the procedure is inserted into the library file. See Figure 3-7.

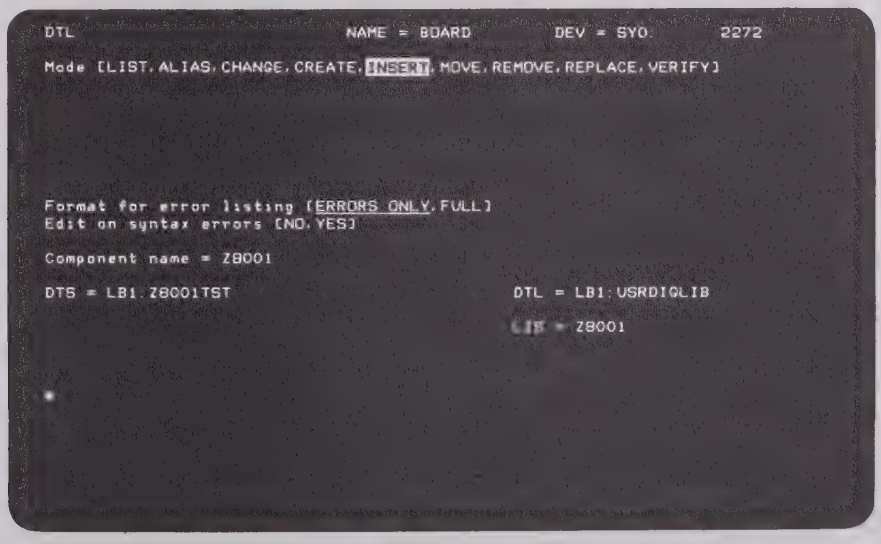

DTL NAME = BOARD DEV = SY0: 2272
Mode [LIST, ALIAS, CHANGE, CREATE, INSERT, MOVE, REMOVE, REPLACE, VERIFY]
Format for error listing [ERRORS ONLY, FULL]
Edit on syntax errors [NO, YES]
Component name = Z8001
DTS = LB1:Z8001TST
DTL = LB1:USRDIGLIB
LIS = Z8001

Figure 3-7. Example of DTL options page showing the command to insert a procedure into a user .DTL file.

Analog Components

The INSERT task for analog components performs a compilation of the component description to a working file format and provides compile error messages. If there are no errors, the working description is inserted into the library file.

3.3 MODIFYING EXISTING LIBRARY FILES

3.3.1 General

Once a User or Update library exists it may be modified by:

1. Replacing an item with one in a source file.
2. Adding an item from a source file.
3. Deleting an item or deleting the entire library (except skeleton).
4. Copying an item from another library.

3.3.2 Replacing Item in Library

An item may be replaced in a User or Update library by providing the new item in a source file. The source file is named in the LBS =, DTS =, or ACS = option field, the library is named in the LIB =, DTL =, or ACL = option field, and execution is selected with the REPLACE mode command. In the case of analog library management, ATL will look at each item in the named source file and will search the named library for a corresponding item. Each time a match is found a replace will be executed. Each item is translated from ASCII to ATG format as it is inserted. In the case of digital or analog component library management, the source file may contain only the one item for each replace operation.

3.3.3 Adding Item to Library

An item may be added to a User or Update library by providing the new item in a source file. The source file is named the LBS =, DTS =, or ACS = option, the library is named in the LIB =, DTL =, or ACL = option, and execution is selected with the INSERT mode command option. In the case of analog library management, ATG will look at the name of each item in the named source file and will search the named library for a matching name. Each time there is no match, the item is inserted into the library. Each item is translated from ASCII to ATG format as it is inserted. In the case of digital or analog component library management, the source file may contain only the one item for each insert operation.

3.3.4 Deleting Item from Library

An item may be deleted from a User or Update library. The library must be named in the LIB =, DTL =, or ACL = option. The type of item being deleted is indicated in the "Section of library" option, for analog. The name of the item is indicated by the "Test or directory name =" or "Component name =" option. The REMOVE mode command option selects the deletion.

3.3.5 Copying Item from One Library to Another

An item may be copied from an analog or digital library to the User or Update library. The originating library is named in the LIBorigin = or DTLorigin = option and the destination library in the LIB = or DTL = option. The type of item being copied is indicated in the "Section of library" option, for analog. The name of the item is indicated by the "Test or directory name ="

or "Component name =" option. The MOVE mode command option selects execution of the copying. ATL will check to see that a matching item does not already exist in the destination library before the copy is made. No translation is needed for this insert.

3.4 LISTING A LIBRARY FILE

3.4.1 General

A listing can be made of various portions of any translated library file, any digital library file, and any compiled analog component library file. The listing process is different for each type of library. The parts of these libraries that can be listed are:

The outline of an analog library
The complete directory of any type library
A characteristic directory of an analog library
A partial directory that can be specified with a common character set and wild-card characters from any type of library
A test procedure from an analog or digital library
A component description from an analog component library
A pin-list for a component in a digital library

In the case of an analog library, the translated file is not readable, so a de-translation must be done before writing the list file. In the case of a digital library, the procedures are in the same form as the source so the list file is copied directly from the library. The DTL monitor page must be used, however, when listing or editing a library file, because of data pointers that DTL puts in the file. The analog component library contains both the compiled description and the source description, so the list file can be copied directly from the library. The procedure for listing some library elements is provided in the following paragraphs.

3.4.2 List a Directory from the Analog User Library

An example of how the options page is completed when listing an analog library directory is shown in Figure 3-8.

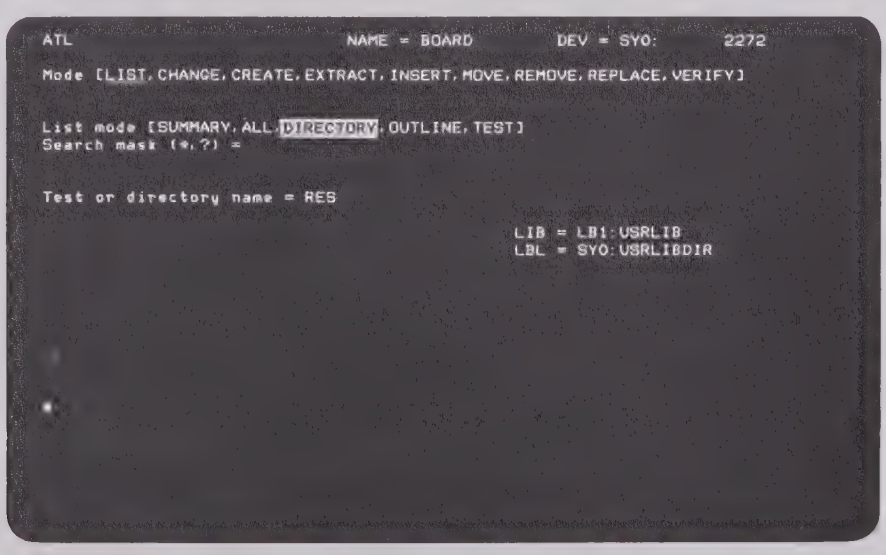

ATL
NAME = BOARD
DEV = SY0:
2272
Mode [LIST, CHANGE, CREATE, EXTRACT, INSERT, MOVE, REMOVE, REPLACE, VERIFY]
List mode [SUMMARY, ALL, DIRECTORY, OUTLINE, TEST]
Search mask (*,?) =
Test or directory name = RES
LIB = LB1:USRLIB
LBL = SY0:USRLIBDIR

Figure 3-8. Example of Options Page for Listing Directory.

3.4.3 List the Entire System Analog Library

The format of a listing is different from a library source since the comments that were in the original source are missing because they were ignored during translation.

An example of the source library is shown in Figure 3-9. This figure contains the Outline and Directory sections of the library and portions of the 2TC Test.

```
$OUTLINE{
C OPENS;        ! MULTI-NAIL NODE CONTINUITY TEST
C NOPEN;        ! ATG JUMPER CONTINUITY TEST
G NOCONT NO;    ! NORMALLY OPEN TEST FOR CONTACTS
G NOCONT OJ;    ! NORMALLY OPEN TEST FOR JUMPERS
G CONT NC;      ! CONTINUITY TEST FOR NC CONTACTS
G CONT J;       ! CONTINUITY TEST FOR JUMPERS
G CONT F;       ! CONTINUITY TEST FOR FUSES
G CONT L;       ! CONTINUITY TEST FOR INDUCTORS
C NOPSHRT;      ! SHORT TEST FOR VCC & GND
C NOSHRT;       ! SHORT TEST FOR ANALOG & DIGITAL NODES
G RES F;
G RES R;
G RES L;
G POT RV;       !PROCESS AFTER OTHER RES TEST FOR PROPER FSUB
G CAP C;
G POL CP;
G TANT CP1;
G IND L;
G CRFORWD CR;
!G LEAKAGE CR;  !UNCOMMENT TO PRODUCE REVERSE DIODE TESTS
G QGAIN QN;
G QGAIN QP;
G ZENER VZ;
G THYRIST SCR;
G TRANCON NJFET;
G TRANCON PJFET;
!G OPAMP LM301A;
!G OPAMP LM301B;
!G OPAMP LM301C;
U TPDEC;
U TPMUX;
U TPEND;
U TPSUB;
}

$DIRECTORY{OPAMP}
$ENTRY{RAIL}
$ENTRY{RAILFOR}

$DIRECTORY{CONT}
$ENTRY{NOTOPEN}

$DIRECTORY{NOCONT}
$ENTRY{ISOPEN}

$DIRECTORY{POT}
$ENTRY{RVFIXG}
$ENTRY{RVADJG}
$ENTRY{POTFIX}
$ENTRY{POTFIXG}
$ENTRY{POTADJ}
$ENTRY{POTADJG}
$ENTRY{PFORM}

$DIRECTORY{RES}
$ENTRY{RDIG}
$ENTRY{RGRD}
$ENTRY{2TR}
$ENTRY{RFORM}

$DIRECTORY{POL}
$ENTRY{CPGRD}
$ENTRY{2TCP}
$ENTRY{CPFORM}

$DIRECTORY{TANT}
$ENTRY{CPGRD}
$ENTRY{2TCP}
$ENTRY{CPFORM}

$DIRECTORY{CAP}
$ENTRY{CGRD}
$ENTRY{2TC}
$ENTRY{CFORM}

$DIRECTORY{IND}
$ENTRY{LGRD}
$ENTRY{2TL}
$ENTRY{LFORM}

$DIRECTORY{CRFORWD}
$ENTRY{CRFWDG}
$ENTRY{CRFWD}
$ENTRY{JUNCTST}
$ENTRY{CRFDFOR}

$DIRECTORY{LEAKAGE}
$ENTRY{CRREVIG}
$ENTRY{CRREVI}
$ENTRY{CRRIFOR}

$DIRECTORY{ZENER}
$ENTRY{VZ}
$ENTRY{VZFORM}

$DIRECTORY{QGAIN}
$ENTRY{QSIM}
$ENTRY{Q}
$ENTRY{QB}
$ENTRY{QFORM}

$DIRECTORY{THYRIST}
$ENTRY{SCRIM}
$ENTRY{SCRVM}
$ENTRY{SCRFORM}

$DIRECTORY{TRANCON}
$ENTRY{JFETFOR}

$TEST{2TC}

$CONFIG{
.TYPE 3,1,2,REF;
.NEED Z[1,2],Z[1,REF],Z[2,REF];
        .IFNDF VALUE;           !NO VALUE -- BAIL OUT
        .QUIT IGNORE NO VALUE;
        .FI;
.IF DBOOL(2);
        .QUIT IGNORE PARALLEL IMPEDANCE TOO LOW (COMP %2N);
.FI;
.ASSIGN CCX=VALUE;
}

$SELECT{
.ASSIGN OME=2*3.14*FS;          !2 F -- W -- OMEGA
?.ASSIGN DISPLAY=B[1,2]/OME;
?.ASSIGN DISPLAY=BX/OME;
.ASSIGN PDERR=B[1,2]/BX;
        .IF PDERR<0.;
        .ASSIGN FLG2=-1.;       !IF NEGATIVE SET FLAG =-1
        .ASSIGN PDERR=-PDERR;   !& MAKE PDERR A POSITIVE
        }

$PARAMETER{
.ASSIGN SKPFLG=0;       ! INITIALIZE OPTION SKIP FLAG
.DEFINE MERR=TSTERR;    !MEASUREMENT ERROR FOR REPORT FILE
.ASSIGN TEMP1=VBIAS;
.ASSIGN CTOT=BSM/OME;           !TOTAL CAPACITANCE
.IF CTOT>10E-6;                 !CHECK FOR CAPS OVER 10UF
}

$FORMAT{% STREAM43 D{%43S}% STREAM45 D{%45S}
%8N:   MEAS % CP F% CS F AT(%1P:%2P)
% DFSKIP D{       % DLY F % V F % F F % BIAS F
}      CMSG='%N' DMSG='%3V' % HI F % LO F % MAX F% MSG N{;}
% MSG D{          PMSG='% MSG H';
}}
```

Figure 3-9. Example of Source Library.

Test Librarian – Section 4

4.1 GENERAL

Operation of the librarians is achieved by selecting options on the appropriate monitor options page. Manipulation of options pages is described in Section 5 of the System Operation manual. A definition of the function of each option on the library options pages is given in the following paragraphs.

4.3 Analog Test Librarian (ATL)
4.4 Digital Test Librarian (DTL)
4.5 Analog Component Librarian (ACL)

4.2 NOTES ON THE MONITOR

NAME =

The monitor is designed so that NAME is a global assignment and is intended to be the board-under-test file name. Board under test has no meaning in library management, so it is best to ignore this option. Since, however, the default feature of the monitor will cause the NAME option to be used for unspecified options, be sure to provide a name for all applicable options. The alternate approach is to use this option for a library name to take advantage of the default feature. Since library names are not used on any other page it is good practice to return the name to its original value after completing the library management session.

DEV =

This is a global option, but it may be useful on this page if several of the library files are on the same device. When a device is specified for this option, the specified device will be the default device for any unspecified library file devices. It may be good practice to return this option specification to its original value after the library management session is finished. This option is ignored if a device is specified in all other applicable options.

4.3 ANALOG TEST LIBRARIAN (ATL)

The ATL options page is accessed by typing ATL on the keyboard while the monitor is in the command mode on any monitor page. A detailed description of the Analog Test Librarian options is contained in the following paragraphs. The entire options page is shown in Figure 4-1; only those options which apply to the selected mode, however, are actually displayed on the screen. The file specification options appear as the default extension for the applicable file.

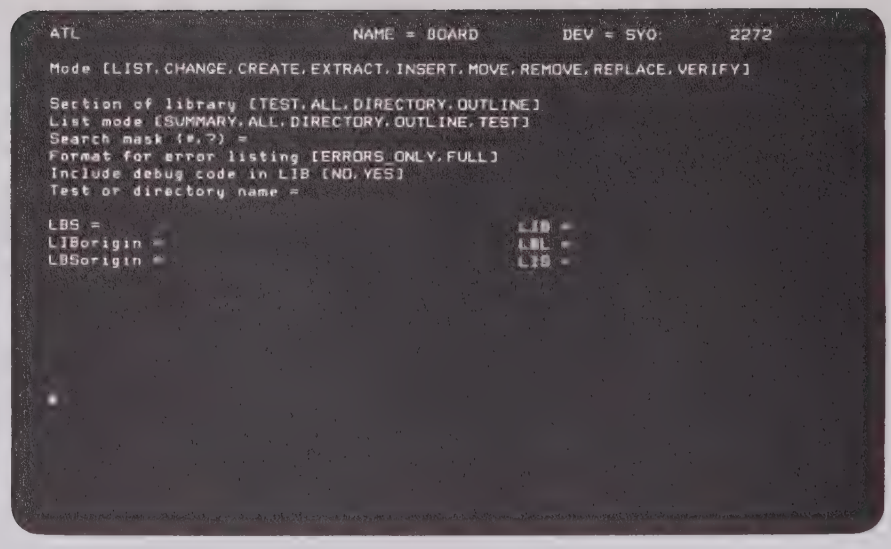


Figure 4-1. Analog Test Librarian Options Page.

ATL Management Options

Mode [LIST,CHANGE,CREATE,EXTRACT,INSERT,MOVE,REMOVE,REPLACE,VERIFY]
Nine library management functions, selected in the "Mode" option, can be performed on analog library files.

LIST Command to de-translate and write out that portion of a translated library file specified in the "List mode" option. The output file is specified by LBL = and can be written to the video display (TI:), a printer (LP0: or LP1:), or a new disk file. The library being listed is specified by LIB = and the file being created is specified by LBL =. .LIB and .LBL are the default extensions for these files.

CHANGE Command to invoke the editor for the purpose of modifying an existing source procedure file. The source file to be modified is specified by LBSorigin = and the new modified source file is specified by LBS =.

CREATE Command to format a new library file so that library procedures can be inserted into it. This command is used only when an Update or User library file is to be initiated. The new User or Update library file is specified in the LIB = option.

EXTRACT Command to copy a test procedure, a directory, the outline, or the entire file (as selected by "Section of Library" option) from an existing library source file to a new library source file. The extracted item is not deleted from the original file. The source file from which the item is to be removed is specified by LBSorigin = and the source file into which the item is to be inserted is specified by LBS =. If the item is a test procedure or a directory, the name of the item to be extracted is specified by "Test or directory name =."

INSERT Command to add all items contained in a test procedure source file to a translated User or Update library file. If an item already exists in the User or Update file, it will not be added. The source file is specified by LBS =, the translated library file is specified by LIB =, and the error listing file is specified by LIS =. Translation of the source

file and verification of syntax are performed as part of this command. The type of error listing file is controlled by the "Format for error listing" option. Debug code is inserted by the translator if selected by the "Include debug code in LIB" option.

MOVE Command to copy a test procedure, a test directory, the outline, or the entire file from one translated library file to another. The item is not deleted from the original file.

REMOVE Command to delete a test procedure, a test directory, the outline, or the entire file from the user-translated library file. This command is not allowed on the system library and is not recommended for use on the Update library. The order of ATG library searching makes it unnecessary to delete from the system or Update libraries items that exist in the User library. The User library is specified by the LIB = option. If the item to be removed is a test or a directory, the name of the item is specified in the "Test or directory name =" option.

REPLACE Command to replace all the items in translated User or Update library file with items in a source file that match. Tests or directories in the source file that do not have matching names in the translated library are ignored. The source file is specified by LBS =, the translated library file by LIB =, and the error listing file by LIS =. Translation of the source and verification of syntax are performed as part of this command. The type of error listing file is controlled by the "Format for error listing" option. Debug code is inserted by the translator, if selected by the "Include debug code in LIB" option.

VERIFY Command to translate a source file for the purpose of finding any errors that may exist. No library file output is generated. The source file is specified by LBS = and the error listing file by LIS =. The type of error listing file is selected by the "Format for error listing" option.

Section of library [TEST,ALL,DIRECTORY,OUTLINE]

Option used to select the type of items that are the subject of an EXTRACT, MOVE or REMOVE command.

TEST A test procedure that is one of the component tests in a source or translated library file. For example, the guarded resistance test procedure is in the library file under the title $TEST {RGRD}. The test name RGRD is entered in the "Test or directory name =" option in this example.

ALL Entire library file such as SY0:LAMPTST.LBS or LB1:USRLIB701.LIB. In the case of REMOVE on LIB file, the format is also deleted so a new CREATE command would have to be performed to re-construct the REMOVEd file.

DIRECTORY A directory for a type of characteristic test that is contained in a library file, such as RES for the directory of resistance tests.

OUTLINE The outline that exists in a library file.

List Mode [SUMMARY,ALL,DIRECTORY,OUTLINE,TEST]
Option used to select the type of listing created by the LIST command.

SUMMARY Includes the outline and all the directories contained in a library file.

ALL Includes the entire library file.

DIRECTORY One of the characteristic directories in a library file.

OUTLINE The outline in the library file.

TEST One of the test procedures in a library file.

Search Mask (*,?) =
A search mask can be used when listing directories or tests. This allows several items with similar names to be listed with one command. The asterisk (*) means that any combination of characters may be used in this location. The ? means that any single character may be used in this position.

Examples of search masks

1. *FORM Lists all FORM tests. The names of all FORM tests end in FORM, preceded by a varying number of characters.

2. POT???G Lists the guarded fixed potentiometer and guarded adjustable potentiometer tests. The names for both of these tests have 7 characters, with only the 4th, 5th, and 6th differing.

Format for error listing [ERRORS_ONLY,FULL]
An error listing is created when a source is being translated, as when the INSERT, REPLACE or VERIFY command is being executed. This file can consist of just the error messages or it can be a full source listing with error messages.

Include debug code in LIB[NO,YES]
A source file may contain code which is provided for debugging purposes. The translator can be told to ignore this debug code (a statement preceded by a ?) or to insert it into the translated file.

Test or directory name =
Provides for specifying a test or directory name when a LIST, EXTRACT, MOVE, or REMOVE command is run with TEST or DIRECTORY option selected.

LBS =
Option that appears on the display when a source library must be specified. It is the receiving file in an EXTRACT command, the edited file in a CHANGE command, the source file in a REPLACE, INSERT or VERIFY command.

LIBorigin =
Option that appears on the display when a MOVE command is selected. It is used to specify the original translated file of the item being copied.

LBSorigin =
Option that appears on the display when an EXTRACT or CHANGE command is selected. It is used to specify the original source file of the item being

copied.

LIB =
Option that appears on the display when a translated library must be specified. It is the receiving file in a MOVE, REPLACE, INSERT, or CREATE command and is the source file in a LIST or REMOVE command.

LBL=
Option that appears on the display when the LIST command is selected. It specifies the file to which the listing goes.

LIS =
Option that appears on the display when an error listing file must be specified, as with the INSERT, REPLACE, and VERIFY commands.

4.4 DIGITAL TEST LIBRARIAN (DTL)

The DTL options page is accessed by typing DTL on the keyboard while the monitor is in the command mode on any monitor page. A detailed description of the Digital Test Library options page (Figure 4-2) and the purpose of each option are contained in the following paragraphs.

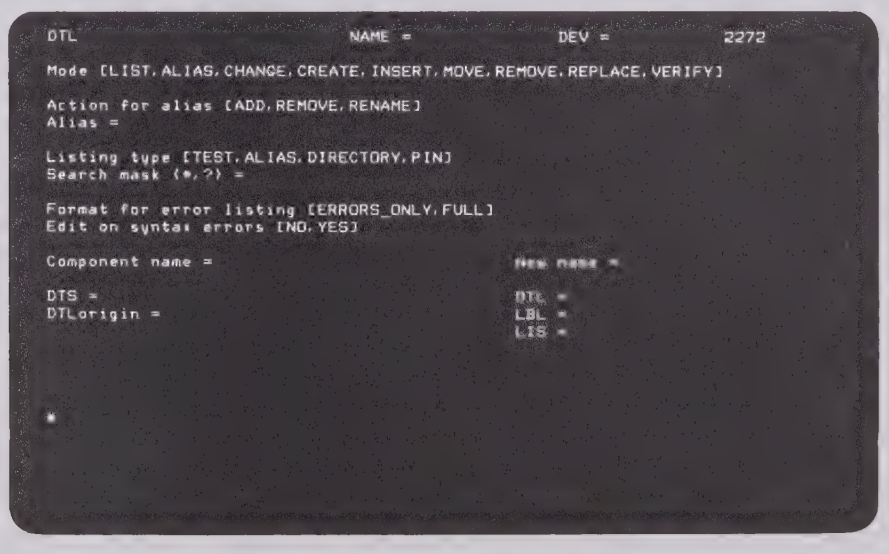


Figure 4-2. Digital Test Librarian Options Page.

DTL Management Options

Mode [LIST,ALIAS,CHANGE,CREATE,INSERT,MOVE,REMOVE,REPLACE,VERIFY]
Nine functions, selected in the "mode" option, can be performed on digital library files. For each mode, the library file is specified by DTL =.

LIST Command to write a portion of a library file to the video display, printer or a new disk file. The portion of the library that is written is selected the "Listing type" option. The filespec for the listing is specified by LBL =.

ALIAS Command to modify the alias list for an IC test procedure. The type of modification is selected by the "Action for alias" option. The alias is specified by "Alias = " and the prime IC type by "Component name =."

CHANGE Command to enter an editing routine to modify a procedure. This routine is a loop that consists of writing the procedure to a new temporary disk file, invoking the editor for creating an edited version of the temporary file, verifying the edited file, replacing the original procedure with the edited file, and deleting the 2 versions of the temporary file. Each of the tasks in this routine is run automatically when the previous task is complete. If there is a syntax error, the routine is halted so that the user can re-invoke the editor and complete the routine manually. If the "Edit on syntax errors" is selected, the editor is re-invoked automatically. Any time there is a syntax error, the error message will be displayed before the system re-enters the edit portion of the loop. Error messages are also written to the error listing file specified by LIS =. The contents of the error listing file are selected by the "Format for error listing" option except when the edit-on-errors option is selected, which automatically creates a FULL listing. The name of the procedure to be changed is specified by "Component name =" and the library file where it is located is specified by DTL =.

CREATE Command to create a new skeleton library file into which procedures can be inserted. This is necessary when starting a new User or Update library.

INSERT Command to add a procedure to a User or Update library file. The procedure must exist in a source file which contains only that one procedure and which is specified by DTS =. The INSERT task includes the verification of syntax which creates an error listing file that is selected by LIS =. The contents of the error file are selected by "Format for error listing" except when "Edit on syntax errors" is selected which automatically creates a FULL listing. If there are any errors, the INSERT task is halted, an error message appears on the display and control returns to the monitor, or to edit-on-errors if selected. The Update or User library file is specified by DTL =. If there is already a procedure name in the library that matches the new name, a message will be displayed and the task will be stopped.

MOVE Command to copy a procedure from one library file to another. The procedure is not deleted from the original library file. If the receiving library has a procedure with the name of the one being moved, a message is displayed and the task is stopped. The original library file is specified by DTLorigin = and the receiving library file by DTL =.

REMOVE Command to delete a procedure from a User or Update library file. The procedure is specified by "Component name =". The primary name of the procedure must be specified. All alternate names are also deleted.

REPLACE Command to replace a procedure in a User or Update library file with a procedure in a source file. The source file must contain only the one procedure for the replacement; this file is specified by DTS =. The REPLACE task includes a syntax checker which creates an error listing as selected by "Format for error listing." When "Edit on syntax errors" is selected, however, a FULL listing is automatically created. The receiving library file is specified by DTL =.

VERIFY Command to check the syntax of a source procedure and create an error listing file specified by LIS =. The contents of the error listing is selected by "Format for error listing." When "Edit on syntax errors" is selected, however, a FULL listing is automatically created. The procedure

file is specified by DTS =.

Action for alias [ADD,REMOVE,RENAME]
This option selects the action that is taken by the ALIAS command. A new alias name is added to the list of aliases for a procedure when ADD is selected. An alias name is removed from the list of aliases when REMOVE is selected, or renamed with RENAME. The alias name being added or removed is specified by "Alias =." The procedure's name is specified by "Component name =;" the procedure's library file is specified by DTL =. A new name for an alias in specified by "New name =."

Alias =
Specifies the alias name in the "Action for alias" options.

Listing type [TEST,ALIAS,DIRECTORY,PIN]
Selects the contents of a file that is created by the LIST command.

TEST A test procedure is written which has the name specified by "Component name =."

ALIAS A list of aliases for the test procedure that is specified by "Component name =" is written.

DIRECTORY A list of all test procedure names in the library is written if "Search mask" is blank. If a name or search mask is specified, all the applicable names will be written.

PIN A listing that indicates the total number of pins and which lists the input pins and the output pins is written for the component specified by "Component name =."

Search Mask (*,?) =
Used to specify a group of names in a LIST/DIRECTORY command. The common character set of the group is entered with a ? used to represent any single letter or digit and an * used to represent any combination of letters or digits. The default for this option is *.

Examples of search masks

1. 74* Lists all IC test names of any length that have 74 as the first two characters.

2. 74?? Lists all IC test names that are four characters long, with the first two characters being 74.

3. ??? Lists all IC test names that have 3 characters.

4. ??LS* Lists all IC test names that have any first 2 characters, then the characters LS, followed by any number of characters.

5. <NULL> Lists the entire directory.

The * only makes sense as the last character of an SEA entry because, once the * is encountered, any remaining characters are ignored.

Format for error listing [ERRORS_ONLY,FULL]
Selects the contents of the error listing file created when a syntax check

is performed. A syntax check is part of the CHANGE, INSERT, REPLACE, and VERIFY commands. The entire procedure with error messages added is written when FULL is selected. Just the error messages are written when ERRORS_ONLY is selected. This option is ignored, however, if "Edit on syntax errors" is selected;in this case, a FULL listing is automatically created.

Edit on syntax errors [NO,YES]
When an INSERT, CHANGE, REPLACE, or VERIFY command is run, a syntax check is performed. Errors must be corrected with the editor. With "Edit on syntax errors," the editor is automatically invoked and, after exiting, the syntax check will automatically be re-run. This loop continues until no errors are found. To abort the entire command prior to obtaining an error-free procedure, when the editor is running type ESC E in TECO and PF1 CTRL E in ED2; the previous version of the error listing (.LIS) is saved. When "Edit on syntax errors" is active, the latest .LIS file is invoked to the editor.

Component name =
Name of test procedure that is the subject of all commands except CREATE and LIST/DIRECTORY.

DTS =
Appears on the display when a source procedure file must be specified. It is used to specify the source in an INSERT, REPLACE, and VERIFY command.

DTLorigin =
Appears on the display so that an original library can be specified in a MOVE command.

New name =
Option for specifying a new alias name to replace the present name.

DTL =
Appears on the display when a library file must be specified. It is used to specify the library in all commands except VERIFY. It is the receiving library in the MOVE command.

LBL =
Appears on the display so that a library listing file can be specified in a LIST command.

LIS =
Appears on the display when a syntax check is to be done so that the error listing file can be specified. It is used in the CHANGE, INSERT, REPLACE, and VERIFY commands.

4.5 ANALOG COMPONENT LIBRARIAN (ACL)

The ACL options page is accessed by typing ACL on the keyboard while the monitor is in the command mode on any monitor page. A detailed description of the Analog Component Librarian options page, Figure 4-3, is contained in the following paragraphs.

The comments below apply in general to the use of the ACL options page.

Any time a function operates on a source file, that file's device and name must be specified in the ACS = option field or the global name that appears

in "NAME" will be used.

Any time a function operates on a library file, that file's device and name must be specified in the ACL = option field or the global name that appears in "NAME" will be used.

Any time a function operates on a component description or alias, the component name must be specified in the "Component name =" field.

Any time a function involves a transfer from a source file to a library file, a file conversion process is performed.

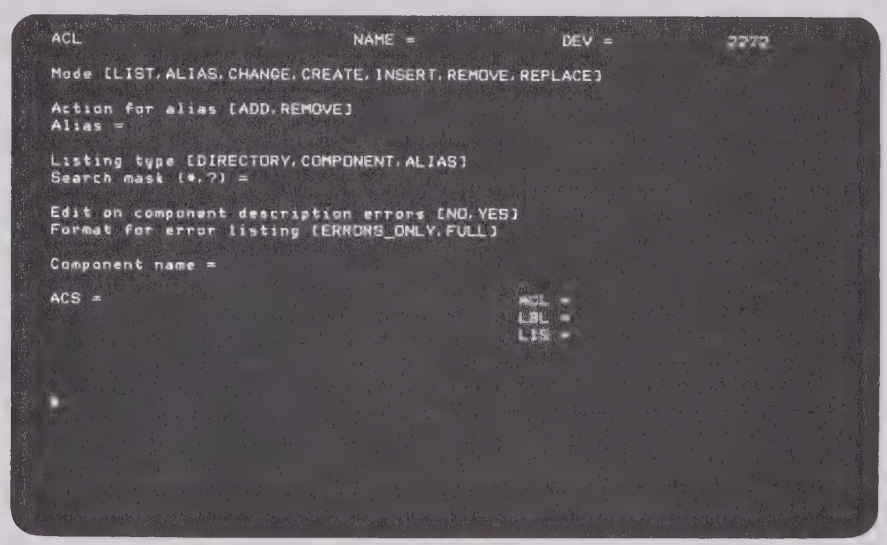


Figure 4-3. Analog Component Librarian Options Page.

ACL Management Options

Mode [LIST,ALIAS,CHANGE,CREATE,INSERT,REMOVE,REPLACE]
Seven functions, selected in the "Mode" option, can be performed on analog component library files. For each mode, the library file is specified by ACL =.

LIST Command to write a listing of a component description, a list of all components in a library, or a list of aliases for a component. The output file is specified by LBL = and can be written to the video display, a printer, or a new disk file. The type of items for the listing is selected by the "Listing type" option. If the type is COMPONENT or ALIAS, the name of of the listed item is specified by "Component name =." If the type is DIRECTORY, the listing will contain all the names represented by "Search mask."

ALIAS Command to add or delete an alias. Select the action in the "Action for alias" option. The alias name to be added or deleted is specified by "Alias =" and the prime name for the procedure by "Component name =."

CHANGE Command to enter an editing routine to modify a component description. This routine is a loop that consists of writing the description to a new temporary file, verifying the edited file, replacing the original description with the edited file, and deleting the 2 versions of the temporary file. Each of the tasks in this routine is run automatically when the pre-

vious task is complete. If there is a syntax error, the routine is halted so that the user can re-invoke the editor and complete the routine manually. The editor is re-invoked automatically if the "Edit on syntax errors" option is selected. Any time there is a syntax error, the error will be displayed before returning to the monitor or entering the edit-on-errors loop. Error messages are also written to the error listing file specified by "LIS." The contents of the error listing file are selected by the "Format for error listing" option except when edit-on-errors is selected, which automatically creates a FULL listing. The name of description to be changed is specified by "Component name=" and the library file where it is located is specified by ACL =. TI: cannot be selected for LIS when edit-on-errors is active.

CREATE Command to create a new skeleton library file into which compiled component descriptions can be inserted. The function is necessary when starting a new User or Update library file.

INSERT Command to compile the component description that exists in a source file and to insert it into a User or Update library file. The source file must contain only a single description which is specified by "Component name =." If a description with the same name already exists in the library, this command is aborted after a message is displayed. The library file is specified by ACL = and the source file by ACS =. "Format for error listing" specifies the contents of the error message file created by the compile. If "Edit on component description errors" is selected, however, the error message file will automatically be a full listing. The error message file is specified by LIS =.

REMOVE Command to delete a component description from a User library. The component is specified by "Component name =".

REPLACE Command to compile the component description that exists in a source file and to replace a description in a User library with it. The source file must contain only a single component description which is specified by "Component name =." A message will be displayed and the command aborted if no description with the same name exists in the library. The library file is specified by ACL = and the source file is specified by ACS =. "Format for error listing" specifies the contents of the error message file created by the compile. If "Edit on component description errors" is selected, however, the error message file will automatically be a full listing. The error message is specified by LIS =. TI: cannot be selected for LIS for edit-on-errors.

Action for alias [ADD,REMOVE]

This option selects the action that is taken by the ALIAS command. A new alias name is added to the list of aliases by an ADD selection. An alias name is removed from the list of aliases by a REMOVE selection. The alias name to be added or removed is specified by "Alias =." The prime component description name is specified by "Component name =" and the User or Update library file by ACL =.

Alias =

Specifies the alias name in the "Action for alias" options.

Listing type [DIRECTORY,COMPONENT,ALIAS]

This option selects the contents of a listing created by the LIST command.

DIRECTORY
A list of all component descriptions in the library specified by ACL = is written to a file specified by LBL = if "Search mask" is left blank. If a name or search mask is specified, all applicable names are listed.

COMPONENT
A component description, specified by "Component name =," is listed from the library specified by ACL =.

ALIAS
Aliases for the component specified by "Component name =" are listed.

Search mask (*) =
Used to specify a group of names in a LIST/DIRECTORY command. The common characters are entered with an asterisk (*) insert where any combination of letters and digits may be used.

Example of a search mask

RPAK* Lists all component names that have RPAK as the first 4 characters.

Edit on component description errors [NO,YES]
When an INSERT, CHANGE, or REPLACE command is run, a component description compile is performed. Any errors must be corrected with the editor. When "Edit on component description errors" is selected, the editor is invoked automatically and, when the editor is exited, the compile task is automatically re-run. This looping will continue until no errors are found. Once there are no errors the original command will be completed. To abort the entire command prior to obtaining an error-free description, when the editor is running type ESC E in TECO and PF1 CTRL E in ED2. Any versions of the edited file created prior to the current editing session will be saved. TI: cannot be selected for LIS.

Format for error listing [ERRORS_ONLY,FULL]
Selects the contents of the error listing file created when a compile is performed. A compile is part of the CHANGE, INSERT, and REPLACE commands. The entire component description with error messages is written when FULL is selected; just the error messages are written when ERRORS_ONLY is selected. This option is ignored, however, if "Edit on syntax errors" is selected; in this case, a FULL listing is automatically written.

Component name =
Name of component description that is the subject of all commands except CREATE and LIST/DIRECTORY.

ACS =
Appears on the display to specify a source description file for an INSERT, REPLACE or CHANGE command.

ACL =
Appears on the display to specify a compiled library for all commands.

LBL =
Appears on the display to specify a listing file in the LIST command.

LIS =
Appears on the display when an error listing file must be specified, as with the CHANGE, INSERT, and REPLACE commands.

Analog Test Library Programming Guide – Section 5

5.1 INTRODUCTION

Analog library programming consists of 3 types of statements.

1. Library organization statements which are preceded by a dollar ($) character. These statements indicate the beginning of each library section.

2. Analog Test Selection Language (ATSL) statements which are preceded by a dot (.) character. These statements are used in the Configuration, Selection and Parameter sections.

3. Format statements which direct the writing of the selected test procedures. These statements are used in the Format section.

The syntax and purposes of the various statements are defined in the following paragraphs under the appropriate headings. Refer to Section 2 for a description of the organization and processing of library routines.

5.2 ANALOG LIBRARY ORGANIZATION STATEMENTS

Each record or procedure within a library file is identified by a keyword. These keywords are preceded by a special character, the $ symbol, to help ATG search for them. The definition of each library organization keyword is contained in the following paragraphs.

---------OUTLINE---------

General

The OUTLINE is a list of test categories that are contained in the library files. ATG will evaluate the components on the circuit board by categories. If a User or Update library is included in the system, and if there are any new DIRECTORY names included in these libraries, then these libraries must include an OUTLINE that is a copy of the System Library OUTLINE edited to include the new DIRECTORYs.

Format

```
$OUTLINE {
outline item;
   ¦
   ¦
outline item;
}
```

where: $OUTLINE Library keyword

outline item test-class dir-code device-type;

test-class	U, C, or G
dir-code	the name of the directory characteristic code;
device-type	any primitive type or Analog Component Library name or alias; used only with G

The system library outline follows as an example.

Example

1. Complete Release 7.4 GenRad-supplied analog library, with explanations.

```
$OUTLINE  {        Start of outline

C  OPENS;          Connectivity routine to check for opens
C  NOPEN;          Connectivity routine to check for not open
G  NOCONT NO;      General routine to check normally open contacts
G  NOCONT OJ;      General routine to check for open jumper
G  CONT NC;        General routine to check normally closed contacts
G  CONT J;         General routine to check for closed jumper
G  CONT F;         General routine to check fuse continuity
G  CONT L;         General routine to check inductor continuity
C  NOPSHRT;        Connectivity routine to check power shorts
C  NOSHRT;         Connectivity routine to check signal shorts
G  RES F;          General routine for resistance test of fuses
G  RES R;          General routine for resistance test of resistors
G  RES L;          General routine for resistance test of inductors
G  POT RV;         General routine for potentiometers
G  CAP C;          General routine for capacitance test of capacitors
G  POL CP;         General routine for capacitance test of polarized capacitors
G  TANT CP1;       General routine for capacitance test of tantalum capacitors
G  IND L;          General routine for inductance test of inductors
G  CRFORWD CR;     General routine for forward voltage tests of diodes
!G LEAKAGE CR;     Optional general routine for leakage test of diodes
G  QGAIN QN;       General routine for gain of NPN transistors
G  QGAIN QP;       General routine for gain of PNP transistors
G  ZENER VZ;       General routine for Zener diodes
G  THYRIST SCR;    General routine for SCRs
G  TRANCON NJFET;  General routine for transconductance of NJFETs
G  TRANCON PJFET;  General routine for transconductance of PJFETs
G  OPAMP UK5;      General routine for opamps
U  TPDEC;          Utility routine for inserting DECLARE statements
U  TPMUX;          Utility routine for inserting MUX statements
U  TPEND;          Utility routine for inserting END statements
U  TPSUB;          Utility routine for inserting subroutines
}                  End of OUTLINE
```

Comments

1. There are 3 types of outline items, denoted by the first character in the item:

   U Utility routine
   C Connectivity routine
   G General In-circuit routine

2. Spaces are required after the type-of-routine indicator (U, C, G) and after characteristic code (POT, RES, etc) in type G items.

3. The test characteristic codes in the G items are DIRECTORY names.

4. The component type of a G item is the same as the type designation listed in a circuit description (.CKT) file.

---------DIRECTORY---------

General

There is a DIRECTORY for each component characteristic that is included in the libraries. Each directory is an ordered list of tests which may be evaluated for each component with that specific characteristic. The search for a DIRECTORY will always start at the highest ordered library.

Format

```
$DIRECTORY {characteristic}
$ENTRY {first test-name}
     |
     |
     |
$ENTRY {last test-name}
```

where: $DIRECTORY Library keyword

characteristic Name of electrical characteristic or of component type tested by the entry

$ENTRY Library keyword

test-name Name of a test in a DIRECTORY

Example

An example of a potentiometer directory taken from the Release 7.4 GenRad-supplied analog library.

```
$DIRECTORY {POT}
$ENTRY {RVFIXG}
$ENTRY {RVADJG}
$ENTRY {POTFIX}
$ENTRY {POTFIXG}
$ENTRY {POTADJ}
$ENTRY {POTADJG}
$ENTRY {PFORM}
```

Comments

1. ATG evaluates all potentiometer testing by considering the tests in the order listed in the DIRECTORY.

2. See the ENTRY discussion that follows for a description of the test names.

---------ENTRY---------

General

Each item in the DIRECTORY is identified as an ENTRY. Each ENTRY directs ATG to the TEST procedures for evaluation of components with that characteristic. The search for a TEST will always start at the highest ordered library.

Format

$ENTRY {test-name}

where: $ENTRY Library keyword

test-name Name of a test in a DIRECTORY

Example

See examples in the preceding discussion of DIRECTORY.

Comments

Table 5-1 lists all the ENTRYs included in the system analog library.

Table 5-1
ANALOG LIBRARY TEST ENTRYs

Directory	Entry	Description
OPAMP	RAIL	Comparator-mode test for opamps.
	RAILFOR	Comparator-mode form test for opamps.
CONT	NOTOPEN	Connectivity test.
NOCONT	ISOPEN	Opens test.
POT	RVFIXG	Guarded test for a potentiometer that is connected as a variable resistor and which is preset to a known setting.
	RVADJG	Guarded test for a potentiometer that is connected as a variable resistor and which must be interactively adjusted at run time.
	POTFIX	Non-guarded test for a potentiometer that is connected as a 3-terminal device and which is preset to a known setting.
	POTFIXG	Guarded test for a potentiometer that is connected as a 3-terminal device and which is preset to a known setting.
	POTADJ	Non-guarded test for a potentiometer that is connected as a 3-terminal device and which must be interactively adjusted at run time.
	POTADJG	Guarded test for a potentiometer that is connected as a 3-terminal device and which must be interactively adjusted at run time.

Table 5-1 (continued)

Directory	Entry	Description
	PFORM	A FORM test for potentiometer for which ATG did not find an acceptable test.
RES	RDIG	Resistance test using digital stimulus and analog measurement.
RES (cont)	RGRD	Guarded resistance test. Used for resistor, fuse resistance, and inductor dc resistance tests.
	2TR	Non-guarded resistance test. Used for resistor and fuse resistance and inductor dc resistance tests.
	RFORM	FORM test for a resistor, fuse or inductor dc resistance for which ATG did not find an acceptable test.
POL/TANT	CPGRD	Guarded capacitance test for polarized capacitors.
	2TCP	Non-guarded capacitance test for polarized capacitors.
	CPFORM	FORM test for polarized capacitors.
CAP	CGRD	Guarded capacitance test for non-polarized capacitors.
	2TC	Non-guarded capacitance test for non-polarized capacitors.
	CFORM	FORM test for non-polarized capacitors.
IND	LGRD	Guarded inductance test.
	2TL	Non-guarded inductance test.
	LFORM	FORM test for inductance.
CRFORWD	CRFWDG	Guarded forward voltage test for diodes.
	CRFWD	Non-guarded forward voltage test for diodes.
	JUNCTST	Basic junction test for diodes.
	CRFDFOR	Form forward voltage test for diodes.
LEAKAGE	CRREVIG	Guarded reverse leakage current test for diodes.
	CRREVI	Non-guarded reverse leakage current test for diodes.
	CRRIFOR	Non-guarded forward current test for diodes.
ZENER	VZ	Zener diode test.
	VZFORM	FORM test for Zener diode.

Table 5-1 (continued)

Directory	Entry	Description
QGAIN	QSIM	Test for transistors with no direct shunt between terminals.
	Q	Test for transistors with direct shunt between terminals.
	QB	Basic transistor test to be evaluated if others fail.
	QFORM	FORM test for transistor.
THYRIST	SCRIM	Switch-action current-measurement test for SCR.
	SCRVM	Switch-action voltage-measurement test for SCR.
	SCRFORM	FORM test for SCR.
TRANCON	JFETFOR	FORM test for JFET.

---------TEST---------

General

Each ENTRY in the DIRECTORY refers to a TEST generation procedure in the library. The actual procedure is identified by the $TEST statement. A TEST must be a complete entity within the library where it exists.

Format

$TEST {test-name}

where: $TEST Library keyword

Comments

1. The format for the TEST statement is similar to that of the ENTRY statement. The test-name in the ENTRY statement must be identical to that in the corresponding TEST statement.

2. See the ENTRY section for a description of all the tests.

---------CONFIGURATION---------

General

Each TEST contains one or more test configurations. ATG sets up a model of a configuration and transfers program control to the SELECTION section for evaluation. This process is repeated for each configuration in the order in which it appears in the CONFIGURATION section, until a satisfactory test is selected or the end is reached.

Format

```
$CONFIG {
      ¦
      ¦
Test Selection Language Statements
      ¦
      ¦
}
```

where: $CONFIG Library keyword

Example

Refer to para 5.4 for a complete description of a CONFIG section.

Comments

1. Test Selection Language statements allowable in the CONFIG section are defined in para 5.5.

---------SELECTION---------

General

The SELECT section of a TEST is a routine to evaluate the test configuration setup in the configuration section. The SELECT statement indicates the beginning of this routine. If a selection is made, program flow will be transferred to the PARAMETER section. If no selection is made, program flow reverts back to the CONFIG section.

Format

```
$SELECT {
     ¦
     ¦
Test Selection Language Statements
     ¦
     ¦
}
```

where: $SELECT Library keyword

Example

Refer to para 5.4 for a complete description of a SELECT section.

Comments

1. Test Selection Language statements allowable in the SELECT section are defined in para 5.5.

2. The $Select stection must be utilized for any G test to succeed.

---------PARAMETERS---------

General

The PARAMETER section of a TEST is a routine to assign values to all the test parameters in the test configuration that was selected in the SELECT section. These parameters will be passed to the FORMAT section where they will be inserted into the test procedure that results from the ATG process.

Format

```
$PARAMETER {
      |
      |
Test Selection Language Statements
      |
      |
}
```

where: $PARAMETER Library keyword

Example

Refer to para 5.4 for a complete description of a PARAMETER section.

Comments

1. Test Selection Language statements allowable in the PARAMETER section are defined in para 5.5.

---------FORMAT---------

General

The FORMAT section of a TEST consists of a test program which includes all the test language statements to be inserted into the final test program produced by ATG. These FORMAT section statements are incomplete. In place of all variable data there is a parameter specification (paramspecs) which directs ATG to the proper values that were defined in the PARAMETER section. There are also key specifications (keyspecs) which provide ATG with information about how, what, and where to write the program statements.

Format

```
$FORMAT {keyspec paramspec {
     |
     |
Test Language statements
}
     |
     |
Test Language statements
}
```

where: $FORMAT Library keyword

keyspec Refer to para 5.6

paramspec Refer to para 5.6

Example

```
1.  $FORMAT {%31S %#NEWTST#D {
        ¦
        ¦
    Test Language Statements
    }
        ¦
        ¦
    Test Language Statements
    }
```

Meaning: $FORMAT { } Defines beginning and end of the FORMAT section.

%31S This keyspec is the sequence or stream indicator that tells ATG where to insert this test into the program. Refer to keyspec description in 5.6.

%#NEWTST#D { } This paramspec consists of a special flagname (NEWTST) which tells ATG to determine if the current component is the first to be tested by the current directory TEST. If true, the D portion of the paramspec tells ATG to insert the Test Language statements that appear between the pair of braces ({ }) into the test program. Refer to paramspec descriptions in 5.6.

Comments

1. The stream keyspec is required to keep all tests that use the same test set-up together so that time will not be lost making unnecessary set-up changes. If the stream keyspec is not included, the default stream (%1S) is assumed and the test will be inserted in the initialization stream.

2. The NEWTST paramspec provides for inserting test initialization statements into the program if the current test is different from the previous test in that stream. If no initial statements are to be used then the paramspec may be left out of the statement.

5.3 ANALOG TEST LIBRARY PROCESSING

The previous paragraphs provided a basic understanding of the purpose for the Analog Test Libraries. Paragraph 5.3 provides a detailed description of how ATG processes the library routines. Figures 5-1 and 5-2 show many of the decisions that are made in finding, accepting and classifying test procedures. Some of the important considerations are:

1. Write the test program so that testing for opens and shorts on the pc board is done before testing components, so that ambiguous results are avoided and so that test time will be saved in case of a board fault.

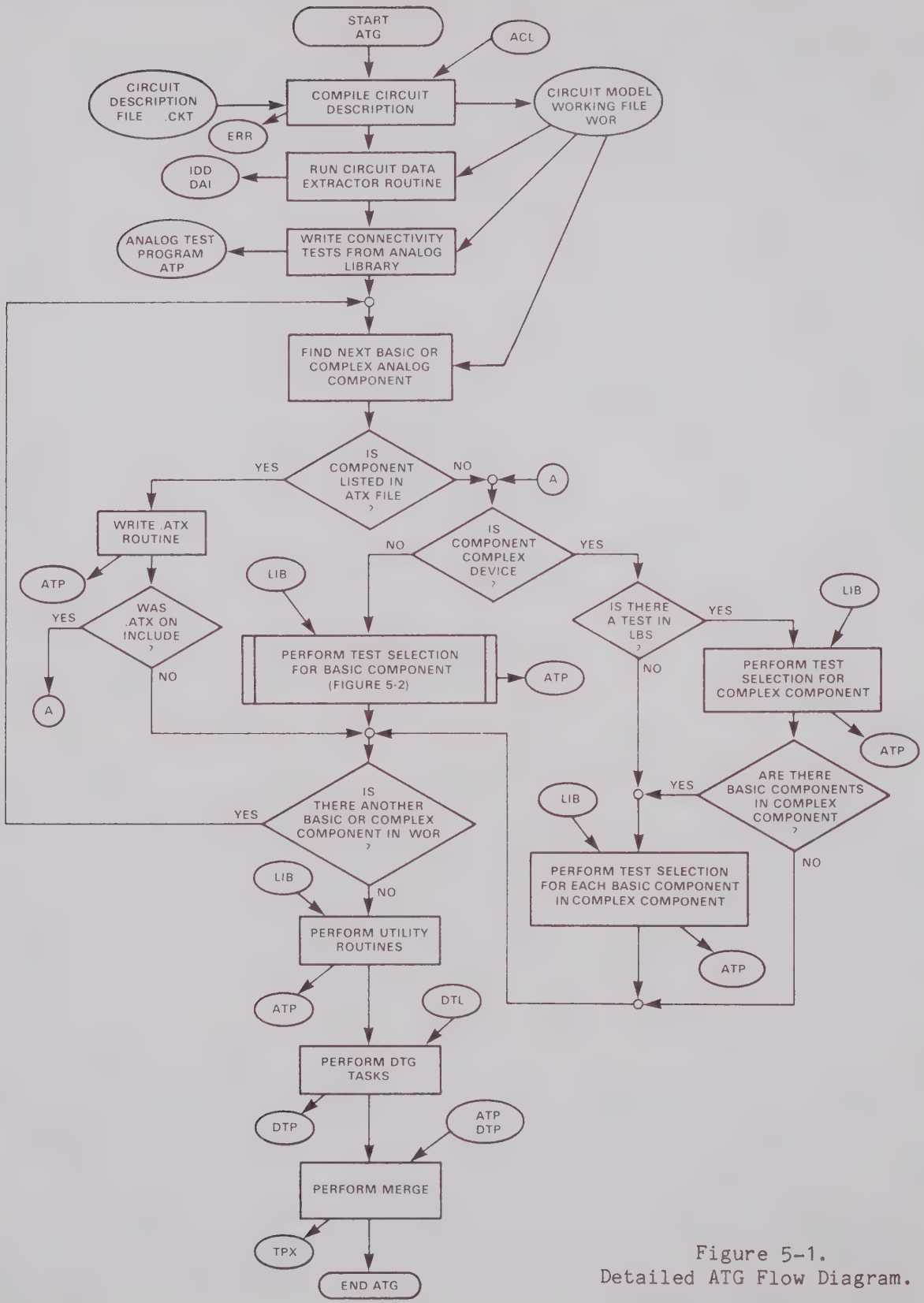


Figure 5-1.
Detailed ATG Flow Diagram.

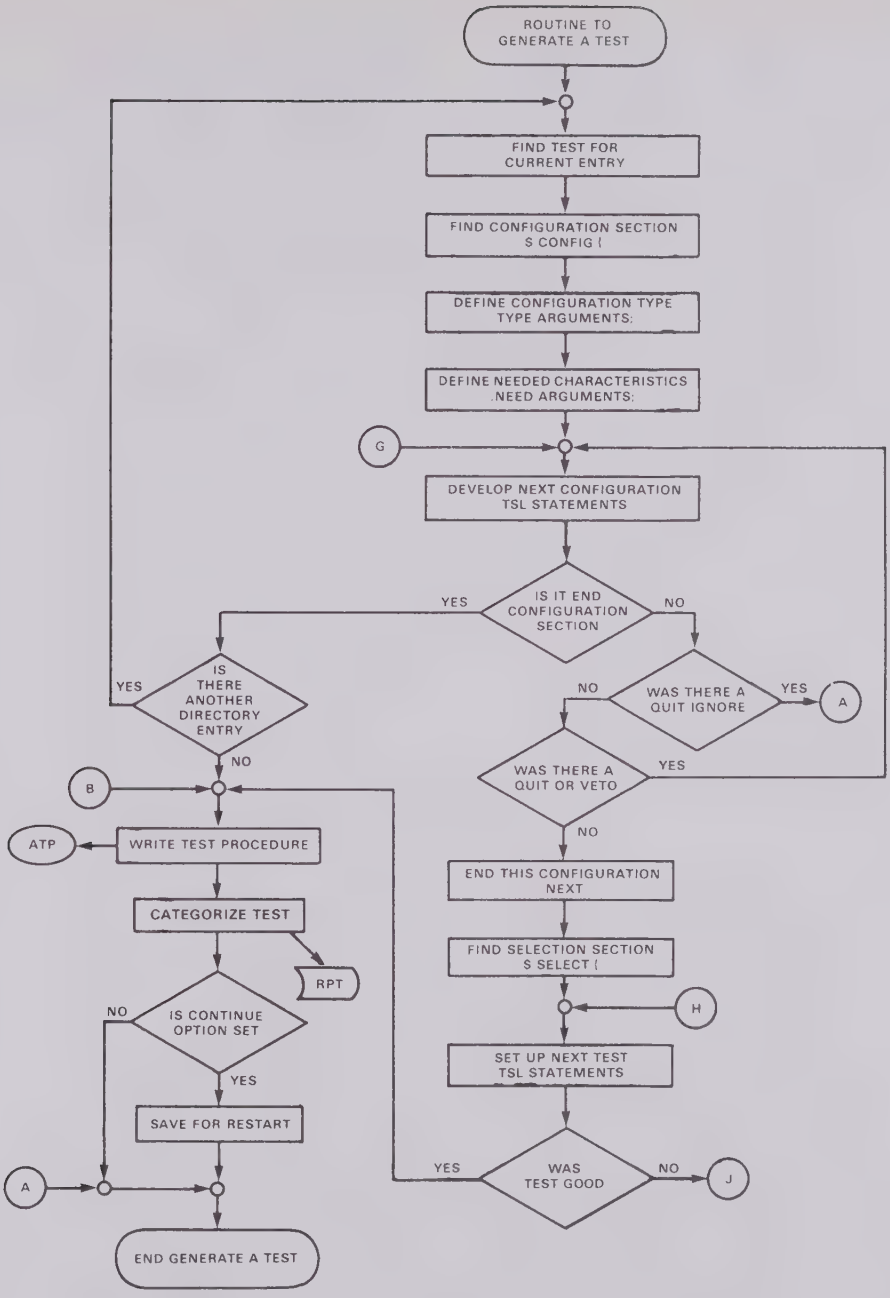


Figure 5-2. Configuration, Selection and Definition of Parameters Flow Diagram (page 1 of 2).

2. Check for discrepancies in the circuit description, such as missing data or repeated names. Decide if a discrepancy is a fatal or a warning situation. Control of processing on warnings is operator selectable.

3. Group test procedures in a way to minimize changes to the hardware, such as MUX and Scanner connections.

4. Was a test with small error limits (superior test) found?

5. Was the best of the tests with larger error limits (adequate test) found?

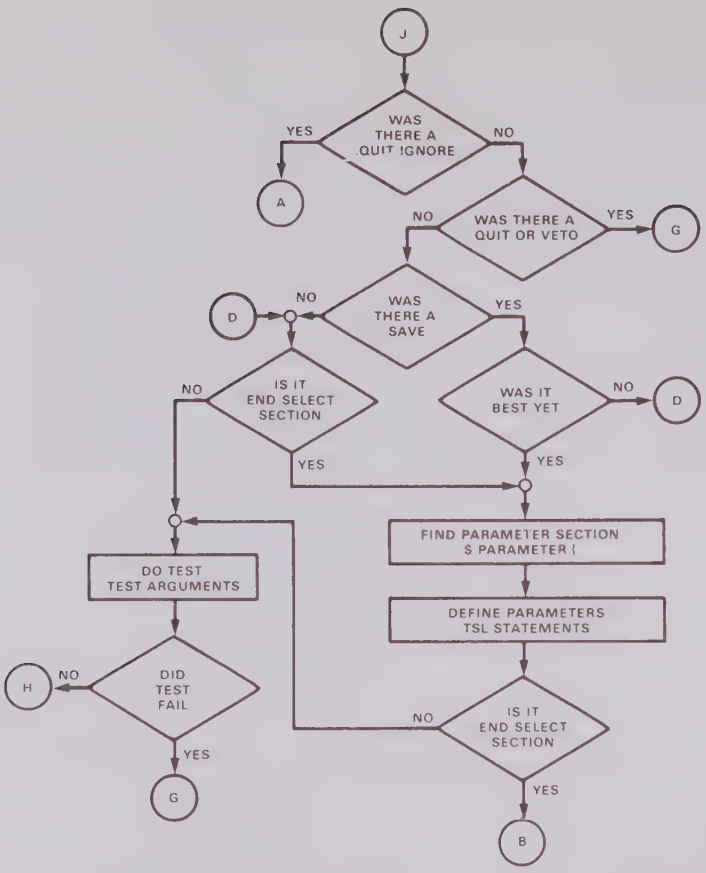


Figure 5-2. Configuration, Selection and Definition of Parameters Flow Diagram (page 2 of 2).

6. Was no test found which had reasonable error limits (unsuccessful test)?

7. Was the component impossible to test (not a testable device) because of circuit environment?

8. Provide a test procedure, called a FORM test, for components which had an unsuccessful test. Add messages to the report file to assist the user in editing the FORM test. This FORM test is inserted into the test program so that it is branched around. If the branch is removed the test can be executed. The FORM test feature is selectable by the user.

User library files may be designed to accomplish a unique objective that is better accomplished by a different set of features. The main point is, if the user can understand why GenRad proceeded the way it did to accomplish the general case, the user can approach the unique case with a better understanding. It is not anticipated that many users will find it advantageous to change significantly the main features of the GenRad library. Minor modification in existing procedures, changes in the order of component testing for more efficient grouping, and addition of special component tests are the primary user concerns.

Evaluation of a test through the CONFIG, SELECT and PARAMETER sections of the library is shown in Figure 5-2. The first statement in the CONFIG section is required to be a TYPE statement. This must be followed by a NEED statement. Then the first measurement configuration is set up by the TSL statements that exist up to the first NEXT statement. Assuming that there were no conditions

that caused a QUIT or a QUIT IGNORE to be exercised, control will be passed to the SELECT section for evaluation of the configuration just set up. The SELECT section consists of a series of tests. If the evaluation proceeds through all the tests with no failures, a good test configuration has been found and control passes to the PARAMETER section for definition of test variables. Then control is passed to the FORMAT section.

If a good test configuration was not found, it may have been because one of the tests failed and control passed back to the CONFIG section. In this case, another measurement configuration would be set up and then the SELECT process would be repeated. If the test configuration was evaluated as adequate, but not good, a SAVE statement would be executed so that the configuration could be used if a better test were not found in a later configuration. Each time a save is executed, control is passed to the PARAMETER section so that all the test variables are defined and saved. If all configurations are evaluated without finding a good test, then control is passed back to the DIRECTORY to look for another TEST for evaluation. A save from a previous TEST will still be available for comparison in the case of another save in the current TEST. If all TESTs in a DIRECTORY, except the last, fail to find a good test, then the last test configuration to be saved will be used for the test program. If no configuration was saved, then the last TEST in the DIRECTORY is processed. If ATG determined that the component is not testable, only enough statements are written to the test program to allow nails to be assigned for the component. If the test may be usable with modifications a 'FORM' test is written. This test has a branch statement at its beginning that would cause it to be skipped during execution. The user will normally edit the test program, adding whatever modifications are required, and then remove the branch statement.

5.4 DEVELOPING AN ANALOG TEST LIBRARY PROCEDURE

This section is designed to lead the user through a logical thought process in the development of an analog component test procedure for the test library. Assume that a simplified, guarded test for a resistor is to be written. Simplified means that many of the considerations actually used in the system library will be overlooked; only primary functions will be considered.

The first step is to draw a diagram of the circuit configurations to be considered. An example is shown in Figure 5-3. A flow chart is then made to show the order of making the calculations and of making decisions about these calculations. See Figure 5-4.

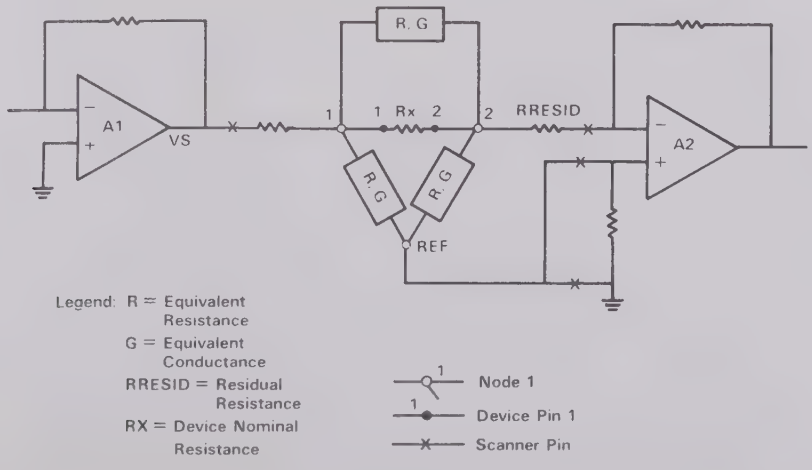


Figure 5-3. Guarded Resistor Test Configuration.

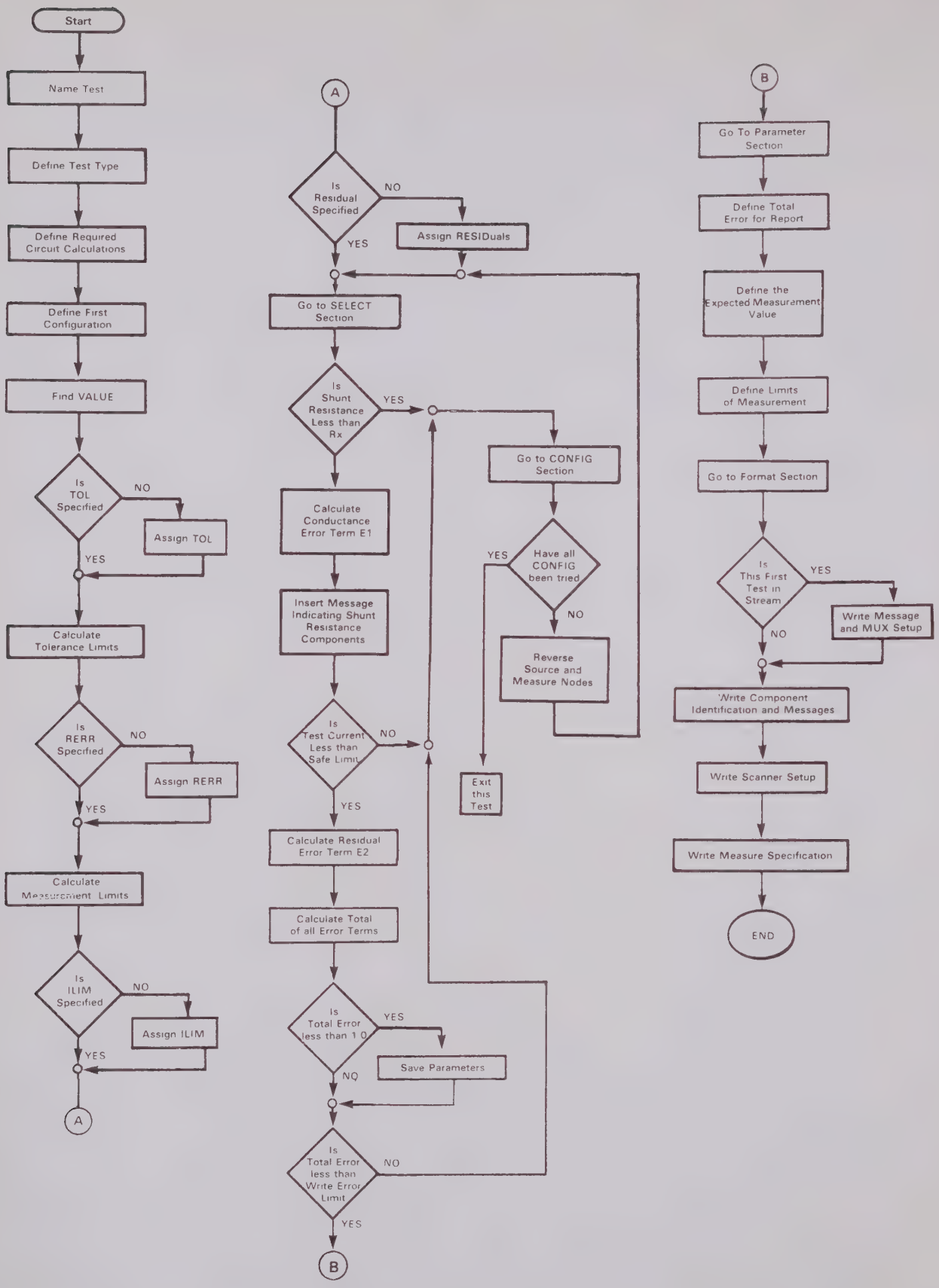


Figure 5-4.
Guarded Resistor Test Flow Chart.

Now the coding of the library test routine can begin. The first line of a test is the label which includes the name of test. If the name of the test is RGRD to signify a guarded resistance test, the label would be

```
$TEST {RGRD}
```

The next line identifies the beginning of the Configuration section of the test.

```
$CONFIG {
```

The next line is the first ATSL statement and must be the statement that defines the type of test. The data for this statement can be taken from the diagram.

```
(1)          .TYPE 3, 1, 2, REF;
```

This tells ATG that there are 3 nodes to be used in the test and that they are called nodes 1, 2 and REF.

This statement is required at the beginning of the test to define the parameters that ATG must calculate before measurement values can be developed. The circuit analyzer portion of the ATG process will model the circuit from the circuit description and calculate the equivalent values specified between the indicated nodes.

```
(2)      .NEED R[1, 2], R[1, REF], R[2,REF], G[1, REF], G[2, REF] G[1, 2];
```

Other statements required near the beginning of each measurement configuration section define the nodes which are to be the source, measure, guard, etc nodes.

```
(3)          .PIN 1 = 1;
```

defines lead 1 of the component connected to node 1 of the circuit. Referring back to the diagram shows that node 1 is the source node. The convention used in the GenRad system library is that node 1 is assigned as the source node for 2-terminal devices.

```
(4)          .PIN 2 = 2;
```

defines lead 2 of the component connected to node 2 of the circuit. Node 2 is the measure node. This is the GenRad system library convention for 2-terminal devices.

```
(5)          .GUARD 1;
```

defines the node to be guarded as node 1.

With the specifying of a source voltage on node 1

```
(6a)         .SET VS = .2;

(6b)         .SET VDC(1) = VS;
```

a measurement configuration is adequately defined so that ATG knows how to make calculations for each specific component that will be evaluated. Statement 6b

will cause node 1 to be at 0.2 volt with respect to both node 2 and the REF node. Thus, the voltage between node 2 and the REF node is zero.

To summarize:

```
$TEST [REF];
$CONFIG 1
.TYPE 1, 1, 2, REF;
.NEED R[1, 2], R[1, REF], R[2, REF], G[1, REF], G[2, REF], G[1, 2];
.PIN 1 = 1;
.PIN 2 = 2;
.SHARD 1;
.SET V2 = .2;
.SET VDC(1) = V2;
```

Next, the specific values associated with a component must be defined. The nominal characteristic value would be defined by

```
(7)         .ASSIGN RES = VALUE;
```

where RES is the variable name chosen by the library programmer to store the value which is obtained from the .WOR file in the VALUE field of the record for that component. The tolerance values would be defined by the following sequence:

```
(8)         .IFNDF HILIM;
(9)         .SET HILIM = RES * 1.05;
(10)        .FI;
(11)        .IFNDF LOLIM;
(12)        .SET LOLIM = RES * 0.95;
(13)        .FI;
```

The interpretation of these statements is: If the upper tolerance has not been specified in the circuit description, define HILIM as nominal plus the default tolerance of 5%; otherwise, HILIM has already been defined from the circuit description. The same thinking applies for the lower tolerance value.

At this point, the measurement error limit concepts discussed in Section 8 of the manual should be referenced. It is necessary to define the Write Test Limits and the Save Test Limits. Therefore, a check for a specification of the flagspec RERR is made, and if not defined in the circuit description, the default value of 1 is used for the measurement error limit calculations. This determination of RERR is:

```
(14)        .IFNDF RERR;
(15)        .ASSIGN RERR = 1.0;
(16)        .FI;
```

and the calculation for the measurement error limits is:

```
(17)        .ASSIGN HER = HILIM-RES;
(18)        .ASSIGN LER = RES-LOLIM;
(19)        .IF HER<LER;
(20)        .ASSIGN ERRLIM = HER/RES*RERR;
(21)        .ELSE;
(22)        .ASSIGN ERRLIM = LER/RES*RERR;
(23)        .FI;
```

The values of tolerance for the upper (HER) and lower (LER) tolerance are calculated so that they can be compared. The smaller of the two values, if one is smaller, is used to calculate the error limit factor (ERRLIM) that will be used for the Write Test Limit. Arithmetic expressions are evaluated on a basis of standard precedence followed by a strictly left to right order, which means that ERRLIM = (HER/RES) X RERR or = (LER/RES) X RERR. Thus, the measurement error limit for a Write Test Condition is a value that is a fraction of the nominal value.

The source current limit and the residual resistance must be defined. Each of these parameters can be controlled by a circuit description flagspec. If no flagspec is defined, the default values of .128 mA for source current limit and 1.8 and 3.2 ohms for low side residual resistance and high side residual resistance are assigned.

```
(24)        .IFNDF ILIM;
(25)        .SET ILIM = .128;
(26)        .FI;
(27)        .IFNDF RLORESID;
(28)        .SET RLORESID = 1.8;
(29)        .FI;
(30)        .IFNDF RHIRESID;
(31)        .SET RHIRESID = 3.2;
(32)        .FI;
```

This completes the configuration statement necessary to define the first configuration of this test. This fact is denoted by

```
(33)        .NEXT;
```

a statement which, in effect, is similar to a jump command. It tells ATG to transfer control to the Select section for evaluation and to return to the next statement in the Configuration section if the test is not selected.

The next 2 statements are all that would be needed to set up a second configuration where the source and measure nodes are reversed.

```
(34)        .PIN 1 = 2;
(35)        .PIN 2 = 1;
```

The source is now on component pin 2, the measure node is component pin 1, and the guard is moved to node 2 because it remains with the source as defined in the first configuration. All configuration statements remain in effect throughout the test unless explicitly changed.

If these are the only configurations to be included in this test, the Configuration section would be ended with the brace

```
            }
```

This tells ATG that there are no more configurations and, therefore, control should go to the next TEST within the current characteristic directory.

The complete Configuration section now consists of the following statements [(1) through (35)]:

```
$CONFIG {
.TYPE 3, 1, 2, REF;
.NEED R[1, 2], R[1, REF], R[2, REF], G[1, REF], G[2, REF], G[1, 2];
.PIN 1 = 1;
.PIN 2 = 2;
.GUARD 1;
.SET VS = .2;
.SET VDC(1) = VS;
.ASSIGN RES = VALUE;
.IFNDF HILIM;
.SET HILIM = RES*1.05;
.FI;
.IFNDF LOLIM;
.SET LOLIM = RES*0.95;
.FI;
.IFNDF RERR;
.ASSIGN RERR = 1.0;
.FI;
.ASSIGN HER = HILIM-RES;
.ASSIGN LER = RES-LOLIM;
.IF HER<LER;
.ASSIGN ERRLIM = HER/RES*RERR;
.ELSE;
.ASSIGN ERRLIM = LER/RES*RERR;
.IFNDF ILIM;
.SET ILIM = .128;
.FI;
.IFNDF RLORESID;
.SET RLORESID = 1.8;
.FI;
.IFNDF RHIRESID;
.SET RHIRESID = 3.2;
.FI;
.NEXT;
.PIN 1 = 2;
.PIN 2 = 1;
}
```

The next statement in this test would tell ATG that this is the beginning of the Selection section.

```
$SELECT {
```

In this section, the library routine tells ATG what calculations to make so that the total expected measurement error can be determined. This expected measurement error is then compared to the Write Test Limit determined in the Configuration section.

First check to determine if the direct shunt resistance is too small.

```
(36)        .TEST RX<R[1, 2];
```

This says to compare whether the nominal value (RX) of the resistor to be tested is smaller than the value of resistance calculated by ATG from the

.NEED statement (R[1, 2]). This is the equivalent resistance between nodes 1 and 2, excluding the component resistance. If the comparison is false, then ATG will exit the Select section and return to the Configuration section statement following the last .NEXT statement that was encountered.

The next check involves determining the expected measurement error (E1) that is due to direct shunt connections which cannot be guarded.

```
(37)        .ASSIGN GMEAS = G[1, 2] + GX;
(38)        .ASSIGN RMEAS = 1./GMEAS;
(39)        .ASSIGN E1 = (GMEAS-GX)/GX;
```

These statements will find a measure of conductance (GMEAS) that is expected between the source and measure nodes. Then this is converted to expected resistance. Finally, this is translated to a number which is a fraction of the component value. This fraction is the measurement error due to parallel conductance. GMEAS is the sum of the conductance between nodes 1 and 2 (G[1, 2]) but not including the conductance of the resistor to be tested (GX). The RMEAS is the reciprocal of GMEAS. E1 is the ratio of the parallel conductance to the conduction of the resistor to be tested. (When parentheses are included in an arithmetic expression, the expression inside the parentheses is evaluated before any operation that follows them. The parentheses in this example do not change the order of evaluation. They are included for user clarity only.) G[1, 2] is the variable name of the parameter that was specified in the .NEED statement.

A message can be inserted at this point which will appear in the report file and which can indicate all the components that are direct shunts.

```
(40)        .SAY VALUE INCLUDES SHUNT RESISTANCE OF %2N;
```

If C35 and R11 were the shunt components, the message that will be inserted into the report file is

```
VALUE INCLUDES SHUNT RESISTANCE OF C35 AND R11.
```

This introduces a convenient feature of ATG which allows variable information to be inserted into the program. There are a number of codes which ATG recognizes as a command to replace the code. Each of these coded commands has the % symbol as the first character. In this case, the code %2N tells ATG to insert a list of all the components which are direct shunts to the device to be tested. These codes are called keyspecs and are defined in para 5.6 of this manual.

Now that a measurement error calculation has been made, it should be tested for a value that is reasonable for a meaningful test.

```
(41)        .TEST (VS/(1.0/(GMEAS + G[1, REF])))<(.85*ILIM);
```

This statement establishes the requirement that the current expected to be drawn from the source must be less than 85% of the current which was calculated to be the maximum current for a valid test. If this condition is not satisfied and no SAVE was executed, program control is immediately returned to the Configuration section statement number 34. Statements 34 and 35 would be executed and then the Selection statements 36 through 41 would be repeated. If the conditions of the TEST statement are satisfied, control will continue to statement 42 to start the evaluation of another measurement error calculation.

The next measurement error term to be considered is the residual error due to instrumentation error and fixture resistances.

```
(42)        .ASSIGN RESIDE = (RHIRESID-RLORESID)/2;
```

This statement assigns a value of resistance to the variable RESIDE. It represents a value which is reasonable for the variability of instrumentation resistance (primarily scanner contact resistance variability). In other words, it is an instrumentation tolerance. This tolerance is converted to an error factor by dividing it by the RMEAS calculated in statement 38.

```
(43)        .ASSIGN E2 = RESIDE/RMEAS;
(44)        .IF E2>ERRLIM;
(45)        .SAY RESIDUAL VARIABILITY CAUSING ERROR;
(46)        .FI;
```

Statement 44 compares the residual error factor to the Write Test Limit (ERRLIM) calculated in statement 20 or 22. If E2 is greater, a message is inserted, by statement 45, into the report file. This will be a likely situation when very small resistors are tested or in the dc test of an inductor.

Many more error terms may be calculated. This example will not show the calculations for other terms, but some of them will be used in the final calculation of the total measurement error for the test. After all the error terms have been calculated, the decision is made to write the test because it has less measurement error than the ERRLIM, to save the test for further consideration after evaluating other measurement configurations, or to ignore the test completely because the measurement error factor is greater than 1.0.

```
(47)        .ASSIGN EM = E1 + E2 + E3 + EGUARD + E5 + E6;
(48)        .IF EM<1.0;
(49)        .SAVE EM;
(50)        .FI;
(51)        .TEST EM<ERRLIM, TOTAL CALCULATED ERRORS EXCEED WRITE TEST LIMIT;
```

These statements add all the measurement errors together and compare the total to 1.0. If EM is less than 1.0, and if ATG finds that it is smaller than any previously saved value, the value of EM will be saved, along with a record that a SAVE was executed. If EM is greater than 1.0, control will jump from statement 48 to statement 51. The error terms of statement 47 could represent the following error considerations:

```
E1 parallel current paths
E2 residual resistance variations
E3 noise signals from large capacitors
EGUARD guard path errors
E4 instrument tolerance
E5 resolution with low current test
```

The SELECT section is ended with a brace.

```
}
```

This tells ATG that there are no more considerations to be made in the SELECT section, which is summarized as follows [(36-51)]:

```
$SELECT  {
.TEST RX<R[1, 2];
.ASSIGN GMEAS = G[1,2] + GX;
.ASSIGN RMEAS = 1./GMEAS;
.ASSIGN E1 = (GMEAS-GX)/GX;
.SAY VALUE INCLUDES SHUNT RESISTANCE OF %2N;
.TEST (VS/(1.0/GMEAS + G[1, REF])))<(.85*ILIM);
.ASSIGN RESIDE = (RHIRESID-RLORESID)/2;
.ASSIGN E2 = RESIDE/RMEAS;
.IF E2>ERRLIM;
.SAY RESIDUAL VARIABILITY CAUSING ERROR;
.FI;
.ASSIGN EM = E1 + E2 + E3 + EGUARD + E5 + E6;
.IF EM<1.0;
.SAVE EM;
.FI;
.TEST EM<ERRLIM, TOTAL CALCULATED ERRORS EXCEED WRITE TEST LIMIT;
}
```

Control will now pass to the next statement, which denotes the beginning of the parameter section.

```
$PARAMETER {
```

This section defines all the parameters, not already defined, that will be used in the test procedure. System variables defined earlier do not have to be redefined. Local variables cannot be used except as defined in the Parameter section. The order of defining parameters is not always important. The following sequence of steps may, therefore, be considered arbitrary. Consider the parameter MERR which has a special meaning. MERR is the parameter, calculated in R, L or C component tests, which is the total expected measurement error.

```
(52)        .DEFINE MERR = EM;
```

This value is inserted into the report in the column headed ERR%.

Parameters needed in the test procedure are defined as follows:

```
(53)        .DEFINE RMEAS = RMEAS;
```

This is necessary because RMEAS is a local variable and therefore must be defined before it can be inserted into the test procedure.

```
(54)        .DEFINE LO = RMEAS - LER;
(55)        .DEFINE MAX = RMEAS + HER;
```

LO and MAX become the lower and upper measurement limits for a component to pass its test at run time.

The PARAMETER section is ended with a brace.

```
}
```

The test selection process will end with all the data for an acceptable test being defined and saved for access by the "Write Program" process. This process is controlled by the next section of the library TEST, the FORMAT section. This section consists primarily of Test Language statements which are processed by ATG to insert variable information that results from the evaluation and selection process of the previous library sections. The start of this section is indicated by

```
$FORMAT {
```

The first group of commands in this section are concerned with functions common to all components which will use the current library TEST procedure. This is significant because the generated test procedures for all components on the board are grouped in the final test program so that all the procedures resulting from a particular library TEST are in a continuous series. Therefore, instrumentation statements and report format statements need only be included once at the beginning of the series. The first statement in the FORMAT section tells ATG to compare the current TEST with the previous TEST and, if they are the same, the group of statements that follow and that are included between the braces ({ }) are to be ignored - not written into the test program. If they are different, they would be considered, as in the following discussions.

In statement 40, the keyspec feature was introduced. This feature and another feature called the paramspec are used extensively in the FORMAT section. Their effect is to tell ATG what or how to write variable information from the ATG process to the test program. A full discussion of these will not be included here but the effects of some of them will be shown as they apply to this library test. See para 5.6 of this manual for detailed information. It is assumed that the reader is familiar with the test language. Refer to the Test Programming manual for review.

```
(56)        %33S%#NEWTST#D {
```

This statement is interpreted by ATG as follows:

% means the next characters are a keyspec code.

33S is a keyspec code, where S indicates a stream number and the stream number is 33, where the stream number is the means of grouping all the test procedures that were generated from the same library TEST together.

%# means the next word is a paramspec code.

NEWTST is a paramspec code that tells ATG to compare this test procedure with the last test procedure used in the currently open stream.

means the next character is a paramspec write code.

D is the paramspec write code that tells ATG to write into the test program all the statements between the next pair of braces if there was not a match resulting from the comparison of stream numbers.

{ is the starting delimiter that defines the statements that are the subject of this statement.

The next statement will be written if this is the first procedure of stream 33.

```
(57)        WRITE 'NOW TESTING RESISTANCE (GUARDED)
            ';
```

This statement tells ATG to insert the character string, between the tick marks, into the Test Program (.ATP). It is thus a heading for the group of guarded resistance test procedures. The new line code must be inserted before the second tick mark, or the printer will not start a new line the next time it is to print. (The WRITE statement is a Test Language statement and is written into the test program as is. Refer to Test Programming Manual for details of this statement.)

The remaining statements are processed for each component that uses this library Test procedure. First, a comment should be written into the test program that identifies the component to which the following procedure applies.

```
(58)        /*%N (%#VALUE#H OHMS)%M*/
```

This statement is written into the test program in the form of a test language comment. ATG inserts the data called for by keyspecs and paramspecs. The keyspec %N tells ATG to insert the name of the current component. %#VALUE tells ATG to insert the numeric value contained in the system variable, VALUE, which was obtained from the circuit description for the nominal value of the component. #H tells ATG to write the VALUE in floating-point format. The keyspec %M tells ATG to insert into this comment all messages that were generated for the test of this component by the ATG process during this $TEST procedure.

The resulting test program statement generated from library statement 59 could be

```
            /* R6 (1000 OHMS)*/
```

and a variety of messages could be inserted after the). Some of these messages would be the ones that may have been generated from statements 40, 45 and 51 of this test.

The next Test Language statement sets up the scanner contacts that connect to the proper nodes for testing the component.

```
(59)        SET SCAN AT (%1P:%2P:%3P:%3P);
```

The 4 flagspecs in this statement tell ATG to insert the adaptor nail designation attached to the indicated node. If node 1 has nail 7, node 2 has nail 15, and node 3 has nail 2, then the resulting test program statement is

```
            SET SCAN AT (7:15:2:2);
```

Next,

```
(60)        MEAS R INTO RVAL V=%4V HI=%#MAX#H%#LO#F MAX=%#RMEAS#H
                        FAIL(1)[CALL FAILSUB (VAL='%#VALUE#H',
                        MVAL=RVAL, LIMS='%#MAX#H AND %#LO#H',
                        NAME='%N', UNIT='OHMS', LOG=LOG,
                        MSG='%MSG#D{%#MSG#H%%NL %%}')];
```

An explanation of the Test Language MEASure statement will not be made here. The following is a list of all the flagspecs used in this statement with an explanation of their effect in the resulting statement that ATG writes into the test program.

%4V — Replace with value of the system variable VS from statement 6.

%#MAX#H — Replace with the value of the local variable MAX, from statement 55, in floating point.

%#LO#H — Replace with the spec name and value of the local variable LO, from statement 54, in floating point.

%#RMEAS#H — Replace with the value of the local variable RMEAS, from statement 53, in floating point.

%#VALUE#H — Replace with the value of the system variable VALUE, from the circuit description, in floating point.

%#LO#H — Replace with the value of the local variable LO, from statement 54, in floating point.

%N — Replace with the name of the component.

%#MSG#D{%#MSG#H%%NL%%} — Replace with the message enclosed in braces if a message was found as a flagspec in the circuit description. The message is a text string and a new line code is included so that the printer will be ready for a new line after printing this text. %NL% is recognized by the the Test Language as a control sequence but ATG would get confused if additional % symbols were not included. %% tells ATG to write the second %.

Assuming that a particular resistor (R12), whose nominal value is 1000, is being evaluated by ATG, the resulting Test Language statement may be

```
MEAS R INTO RVAL V=.2 HI=1095 LO=915 MAX=1010 FAIL (1)
                [CALL FAILSUB (VAL='1000', MVAL=RVAL,
                LIMS='1095 AND 915', NAME='R12',
                UNIT='OHMS',LOG=LOG, MSG='VALUE MAY
                BE CHANGED AT SYSTEM TEST %NL%')];
```

The $FORMAT section and the $TEST end with a

```
}
```

The earlier discussion of library organization has defined the four library functions: Configuration, Selection, Parameters, and Format. The Format section is not written in ATSL. The other three sections do use ATSL statements. Some of the ATSL statements have applications in all three sections while others have applications in only one section. The following is a list of all ATSL command keywords grouped according to application. These statements are defined in para 5.5.2.

Statements with common applications

```
.ASSIGN     .ELSE
.IF         .FI
.IFDEF      .QUIT
.IFNDF      .QUIT IGNORE
.SAY
```

Configuration Statements

```
.GUARD      .PIN
.NEED       .SET
.NEXT       .TYPE
```

Selection Statements

```
.SAVE       .TEST
```

Parameter Statements

```
.DEFINE
```

Operators recognized by ATSL are:

```
.AND.   Logical AND operator
.OR.    Logical OR operator
<       Less than
>       Greater than
==      Equivalency
+       Add
-       Subtract
*       Multiply
/       Divide
=       Assignment operator
```

System keywords reserved for use in the command statements are listed in groups as follows:

Operand Functions

```
ABS       LOGN        NODEMODE  VDC
DBOOL     MAG         PINCNT
EXIST     MAGSQ       SQRT
EXPN      MYTYPE      TBOOL
GBOOL     NAILCNT     VAC
```

System variables

B	G	RX	Y
BX	GX	VALUE	YX
CAP	HILIM	VBIAS	Z
DLY	LOLIM	VS	ZX
ERR	R	X	
FS	REF	XX	

Other reserved words

D Q

5.5.2 Command Statements

The following pages present a complete definition of the Analog Test Selection Language (ATSL). A definition of the words used in defining the ATSL statements is presented as follows:

General Definitions

node	Represents an electrically common path in the circuit. It is defined by ATG, from the circuit description, for each device configuration that the ATG library evaluates. Node names that may be assigned to each node of a test configuration are the numbers from 1 to 19 and the name REF.
local-variable	A variable defined by the user that ATG only recognizes in the library test procedure in which it is defined. ATG predefines all applicable flagspecs (except message flagspecs) each time a library test is initiated. They are therefore considered local-variables.
system-variable	A variable known to ATG which can be referenced throughout the library.
variable	Local-variable or system variable. Combination of alpha and numeric characters and the underline (_). The first character must be alpha and the maximum number of characters is 8.
format-variable	The object of a DEFINE statement.
pin	A device terminal.
nail	A pin in the bed-of-nails adaptor which makes contact to the circuit board etch. This nail is connected to the system instrumentation through the receiver pins.
expression	A function that can be evaluated by ATG. It may be a single variable or constant or any algebraic function that involves addition, subtraction, multiplication or division, or it may consist of any logical function that involves OR, AND, greater than, less than, or equivalency.

The definition of ATSL statements is presented in groups according to library usage and alphabetically within groups. These groups were defined in para 5.5.1.

--------ASSIGN statement--------

General

The ASSIGN statement is used to define a value to a local variable.

Format

```
.ASSIGN local-variable = expression;
   or
.ASSIGN operand-function = expression;
```

where:		
	ASSIGN	Command keyword
	local-variable	Refer to General Definitions
	expression	Refer to General Definitions
	operand-function	CBOOL or DBOOL or GBOOL or TBOOL

Examples

1. .ASSIGN RESID = RHIRESID + RLORESID/2.;

   Meaning: Make the value of the local variable RESID equal to the average value of variables RHIRESID and RLORESID.

2. .ASSIGN DBOOL (2) = 1;

   Meaning: Sets bit 2 of the DBOOL bit string. Refer to para 5.5.3.

Rules and Considerations

1. A local variable must be assigned a value in a statement which precedes the statement in which it is first used.

2. A local variable name cannot be the same as a system variable name.

3. The "=" symbol is used in ATSL to assign the value on the right to the variable on the left.

4. Algebraic expressions are evaluated strictly from left to right except where grouped with parentheses.

5. All TSL statements are terminated by a semicolon (;).

--------DEFINE--------

General

The DEFINE statement is used in the PARAMETER section to provide values to each of the parameters that are to be inserted into the selected test program via the FORMAT section.

Format

.DEFINE format-variable-name = expression;

where:	DEFINE	Command keyword
	format-variable-name	Refer to General Definitions
	expression	An optional entry. Refer to General Definitions

Example

1. .DEFINE HI = RMEAS+HER+((E2+E4+E5+E6)*RMEAS);

   Meaning: The value of the parameter HI is defined to be the value of the algebraic function.

--------GUARD--------

General

The GUARD command is used to tell ATG what terminal in the test configuration is to be guarded. Typically, the node to which one of the terminals of the device under test is connected is the node specified in a GUARD statement. The numbering of nodes in a test configuration is arbitrarily assigned by the library programmer. Usually, a test configuration for a 2-terminal device would be assigned nodes 1, 2 and REF. REF is the GUARD node so would not be specified in the GUARD statement.

Format

.GUARD node;

where:	GUARD	Command keyword
	node	A number from 1 to 10

Example

1. GUARD 1;

   Meaning: Develop a guard set at the test configuration node "1". Node 1 must be defined by previous .PIN statement.

--------IF construct--------

General

The IF construct provides a means of selectively executing groups of statements. It is the only means provided in ATSL for changing the flow of execution within a library section. When the IF condition is true, the statement after the IF is executed next; otherwise the statement after the FInish is executed next. An ELSE statement may also be used in the IF construct. In this case, when the IF condition is true, the statement after the IF is executed next; otherwise the statement after the ELSE is executed next.

Format

```
.IF conditional; or .IFDEF variable name; or .IFNDF variable-name;
      ¦
    TSL statements
      ¦
[.ELSE;]
      ¦
    TSL statements
      ¦
.FI;
```

where: IF — Command keyword

conditional — Consists of expression operator expression

[.ELSE;] — [ ] mean that an ELSE is optional

Examples

1.
```
.IFNDF RHIRESID;
.ASSIGN RHIRESID=3.2;
.FI;
```

Meaning: If the value of the variable RHIRESID was not defined in the circuit description file, then assign the value of 3.2 to it. Otherwise, execute the statement after the .FI. FI indicates the end of the IF construct. RHIRESID is a flagspec that could have been defined in the circuit description. If it is not defined in the circuit description, the default value of 3.2 is assigned.

2.
```
.IF VM>0.5;
.ASSIGN SE=1.3E-3;
.ELSE;
.ASSIGN SE=3.2E-4;
.FI;
```

Meaning: If the value of the variable VM is greater than 0.5, then assign the value of 1.3E-3 to the variable SE. If it is not, assign the value of 3.2E-4 to the variable SE. In this example VM, is a variable name for a voltage measurement calculated by ATG in an earlier statement. SE is a variable name for source error to be assumed by ATG in calculating the measurement errors for a resistor test.

3.
```
.IF IMEAS<39.0E-6;
.ASSIGN RF=64.0E+3;
.ASSIGN RIN=1.0;
     .ELSE;
     .IF IMEAS<625.0E-6;
     .ASSIGN RF=4.0E+3;
     .ASSIGN RIN=0.2;
          .ELSE;
          .IF IMEAS<10.0E-3;
          .ASSIGN RF=250.0;
          .ASSIGN RIN=1.0;
              .ELSE;
              .ASSIGN RF=15.6;
              .ASSIGN RIN=.1;

.FI; .FI; .FI;
```

Meaning: IFs can be nested and there may be one ELSE within an IF construct. There must be a FI for each IF. The first FI ends the last IF sequence, etc. This example is setting the range of the dc measure, depending on the measurement current previously calculated by ATG.

These 3 examples were taken from the non-guarded resistance test in the library.

Rules and Considerations

1. When the conditional is true, the statements immediately following the IF statement are executed.

2. When the conditional is false, the flow of execution is transferred to the next statement after the next FI or the next ELSE.

3. IF constructs can be nested.

4. The IFDEF and IFNDF statements are interpreted "if the variable has been defined/not defined then execute the next series of statements, otherwise, execute the statements following the next FI or ELSE."

5. All TSL statements must be terminated with a semicolon (;).

--------NEED--------

General

The NEED command is used to tell ATG what equivalent circuit characteristics must be calculated for the present test. The calculated characteristics do not include the component being tested.

Format

```
.NEED characteristic [node, node];
```

where: NEED — Command keyword

node — Character string REF or a number from 1 to 10

characteristic — One of the keywords CAP, B, G, R, X, Y, or Z

Example

1. .NEED R[1,2], R[1,REF], R[2,REF];

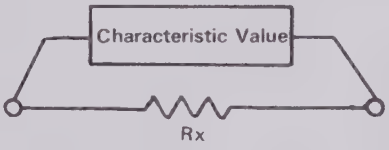
   Meaning: Calculate the equivalent value of resistance between nodes 1 and 2, 1 and REF, and 2 and REF.

Rules and Considerations

1. Each item in the statement becomes a variable name where the calculated value is stored. In this example R[1,2] becomes the variable-name.

2. When REF is used as a node, it must follow the node it is paired with.

3. Characteristic values are defined as follows:

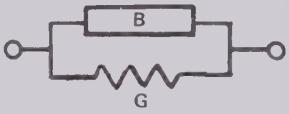


RX = Value of component being tested

For AC Series equivalency

$Z = R + jx$ $\quad R = \frac{G}{G^2 + B^2}$ $\quad X = \frac{-B}{G^2 + B^2}$

For AC Parallel equivalency



$Y = G + jB$ $\quad G = \frac{R}{R^2 + X^2}$ $\quad B = \frac{-X}{R^2 + X^2}$

4. Only one NEED statement is allowed per test. Include all characteristics needed in the same statement.

--------NEXT--------

General

The configuration section of a test may have several configurations which may be evaluated by the selection section of the test. The flow control statement which tells ATG to go to the selection procedure is the NEXT statement. The configuration defined just prior to the NEXT statement will be evaluated by the selection section. If no selection is made, the ATG process flow will return to the statement following the NEXT statement. This would be the first statement of the next configuration. The brace (}) at the end of the configuration section causes the same function as .NEXT;.

Format

.NEXT;

--------PIN--------

General

The PIN statement assigns the pins of the component being tested to a terminal of a test configuration. PIN works in two different ways. When no nail-type-list is entered, ATG is implicitly instructed to keep searching for an acceptable test of the specified configuration. Entering a nail-type-list explicitly requires ATG to enter $SELECTION for the configuration only when the component nail has available a scanner of the type listed.

Format

.PIN [nail-type-list] node = component-pin-name;

where:	PIN	Command keyword
	component-pin-name	Any number or letter allowable in the circuit description file for component pin designators. Typically, component pin names are 1, 2, 3, A, B, C, E, N or P. Refer to System Operation Manual.
	[nail-type-list]	One to 3 of the four nail-types, A, D, H and F. Separate the nail-types with a comma. The "[]" indicate that the nail-type-list is optional in the statement.
	node	See General Definitions.

Example

1. .PIN 2=1;

Meaning: Assign component pin 1 to node 2.

--------QUIT--------

General

The QUIT is used to terminate the evaluation and selection process for the present test and to insert a message into the report file, if desired.

Format

.QUIT [message];

where:	QUIT	Command keyword
	[message]	Any character string. [ ] indicate an optional entry.

Example

1. .IFNDF PRESET;
   .QUIT FIXED TEST NOT APPROPRIATE;

Meaning: If the flagspec PRESET is not defined, quit this fixed pot test. ATG will then go to the adjustable pot test.

--------QUIT IGNORE--------

General

The QUIT IGNORE command is used to terminate the evaluation and selection process for the present test when ATG determines that the component is not testable because of its circuit environment. The not testable statistic is added to the report file.

Format

.QUIT IGNORE [message];

where: QUIT IGNORE Command keyword

[message] Any character string. [ ] indicate an optional entry.

Example

1. .IFNDF DCR;
   .QUIT IGNORE RESISTANCE TEST NOT POSSIBLE WITHOUT DCR SPECIFICATION;

   Meaning: If no flagspec was provided in the circuit description for the dc resistance of an inductor, quit and insert the message into the report file.

--------SAVE--------

General

The SAVE command is used near the end of the SELECT section to store the current test configuration. ATG will then decide whether or not actually to store it, based on the following criteria.

1. It is the first configuration within this test which is to be saved.

2. It has a better figure of merit than the configuration previously saved. In this case, the previously saved test is lost.

Format

.SAVE expression;

where: SAVE Command keyword

expression Refer to General Definitions

Example:

1. .ASSIGN EM = E1+E2+E3+E4+E5+E6;
   .IF EM<1.0;
   .SAVE EM;
   .FI;

   Meaning: Sum up all the error terms previously calculated and store in variable EM. If EM is less than 1.0, then ATG will consider storing the current test configuration, using EM as the figure of merit.

--------SAY--------

General

The SAY command is used to insert a message into the report file.

Format

.SAY message;

where: SAY Command keyword

message Any character string

--------SET--------

General

The SET command is used to define a value for a system variable.

Format

1. .SET system-variable = expression;

where: SET Command keyword

system-variable Refer to General Definitions

expression Refer to General Definitions

2. .SET operand [node, node];

where: operand VDC or VAC

node, node Two nodes between which the voltage is defined. The first node is the specified voltage with respect to the second node.

Example

1. .SET HILIM = RES*1.05;

Meaning: Make the value of the system variable HILIM equal to the previously defined local variable RES multiplied by 1.05.

2. .SET VDC [3,2] = 0.2;

Meaning: A dc voltage is defined between nodes 3 and 2, so that node 3 is 0.2 volt with respect to node 2.

Rules and Considerations

1. SET and ASSIGN statements cannot be used interchangeably. ASSIGN is only for local variables and SET is only for system variables.

--------TEST--------

General

The SELECT section has one or more parameters which must be checked during the selection process. The TEST command is the last statement in each of the routines which form one of these checks. ATG will take note of whether or not the conditions of the TEST statement are true. If true, ATG will continue to the next statement in the SELECT section. Upon a false condition, a message will be inserted into the report file if a message is included in the TEST statement. Then ATG will return to the CONFIG section to the statement following the NEXT statement from which the CONFIG section was last entered. In other words, ATG will go back to set up the next configuration.

Format

.TEST expression comparison-operator expression [, message];

where:	TEST	Command keyword
	expression	Refer to General Definitions
	comparison-operator	A < or > or ==
	[, message]	A character string. [ ] indicate an optional entry.

Example

1. .TEST EM<ERRLIM;

   Meaning: Check to see if EM is less than the previously assigned error limit value ERRLIM. Note a false condition and continue or branch as applicable.

--------TYPE--------

General

The TYPE command is used to declare the node set to be used in the configurations considered by this test.

Format

.TYPE number-of-nodes, node-list;

where:	TYPE	Command keyword
	number-of-nodes	Number from 1 to 20
	node-list	List of node names
	node-name	Character string REF or a number from 1 to 19

Example

1. .TYPE 3,1,2,4;

   Meaning: The test configuration of this test has 3 terminals which are terminals 1, 2, and 4.

5.5.3 System Built-in Functions (Operands)

General

There are a number of functions which may be performed by ATG simply by naming the function within a TSL statement, i.e., by using the appropriate keyword and arguments as the operand of the statement. The available functions are described in the following paragraphs which are arranged in alphabetical order by keyword.

ABS — Makes the resulting variable the absolute value of the expression that follows the keyword.

example: ASSIGN N = ABS (J-K);

meaning: The unsigned value of J minus K will be assigned to N.

example: .IF ABS (J-K)==N;

meaning: If the absolute value of J minus K is equal to N, then execute the next instruction. Otherwise, execute the instruction after the next .FI.

CBOOL — A built-in bit string (bit addresses 1-99) that can be used for flags and which is automatically reset when ATG starts a new component. A component is any device specified in the .CKT file. This bit string is used to group all data for reports on a complex device. The examples given under DBOOL illustrate how Boolean bit strings are used.

DBOOL — A built-in bit string that can be used for flags and which is reset automatically when ATG starts processing a new directory. It is valid for all tests within current directory. The maximum number of bits that may be addressed is 99, which are numbered 1-99.

example: .ASSIGN DBOOL (N) = 1;

meaning: The bit represented by the value of N (where N is an integer constant between 1-99) is set to "1" in the DBOOL bit string.

example: .IF DBOOL (6);

meaning: If bit 6 in the DBOOL string is 1, then execute the next instruction. Otherwise, execute the instruction after the next .FI.

EXIST — Used as an argument in an IF statement. It causes ATG to search the file for a particular type of component. If it is found, the IF condition is considered to be true. The function can also be used to assign a binary value to a variable.

example: .IF EXIST (CP1);

meaning: If there is a polarized tantalum capacitor in the circuit, perform the next instruction. Otherwise, execute the instruction after the next .FI.

example: .ASSIGN C = EXIST (IC);

meaning: If there is a component type IC in the WOR file, the binary value 1 will be assigned to the variable C. If there is no type IC component, C will be assigned the value 0.

EXPN — Makes the resulting variable equal to the evaluated exponential (e^n). The exponent (n) is the argument that follows the keyword EXPN.

example: .ASSIGN K = EXPN (J-2P);

meaning: The value of $e^{(J-2P)}$ is assigned to the variable K.

GBOOL — A built-in bit string (bit addresses 1-99) that can be used for flags that are automatically reset only at the start of a new ATG run. It is global throughout ATG.

LOGN — Makes the resulting variable equal to the natural logarithm of the expression that follows the keyword LOGN. The expression must be greater than zero.

example: .ASSIGN M = LOGN (1/N);

meaning: The value of ln 1/N is assigned to the variable M.

MAG — Make the resulting variable equal to the magnitude of the values of each expression that follows the keyword.

example: .ASSIGN J = MAG (2,N);

meaning: The value of $\sqrt{2^2 + N^2}$ is assigned to J.

MAGSQ — Similar to MAG except that, in the example, J would be assigned the value of $(2^2 + N^2)$.

MYTYPE — Usually used as an argument in an IF statement. It causes ATG to check the present component type and compare it to the type indicated in the statement. If there is a match, the IF condition is considered to be true.

example: .IF MYTYPE (R);

meaning: If the present component being evaluated is a resistor, perform the next instruction.

NAILCNT Usually used as an argument in an IF statement. It causes ATG to search the circuit description for all adaptor nails connected to the indicated node. The quantity of nails found becomes the argument in the statement. In 2270 systems, digital nails are ignored.

example: .IF NAILCNT (2) > 1;

meaning: If the number of nails on node 2 is greater than 1, perform the next instruction.

definition: A nodeset is an entity created by ATG from the circuit description as a result of a NEED, PIN or GUARD statement. It represents an electrically continuous path on the circuit board.

Another form of the NAILCNT operand causes ATG to search circuit description for all nails that are connected to the indicated nodeset. The average quantity of nails per node in the nodeset becomes the argument in the statement.

example: .IF NAILCNT (REF) > 2;

meaning: If the average number of nails per node in the REF nodeset is greater than 2, perform next instruction.

definition: A nodeset is an entity created by ATG from the circuit description as a result of a NEED, PIN or GUARD statement. It represents all the nodes which are connected together to form a node such as a guard node.

NODEMOVE Copies to another node all the nodeset information resulting from a previous .NEED, .GUARD, .PIN or other statement for one node in the present configuration. It also stores in the local variable the number of nodes in the node set. If only the number of nodes is desired, the destination part of the argument is zero.

example: .ASSIGN NEWNODE = NODEMOVE (5,1);

meaning: ATG copies all the nodeset data defined for the node designated as node 1 to the node designated as node 5. This is helpful in equivalent circuit calculations.

PINCNT Assigns to a variable the value that is equal to the number of component pins connected to the indicated node.

example: .ASSIGN PINZ = PINCNT (3);

meaning: Assign the value that is equal to the number of device pins attached to node 3 to the variable PINZ.

SQRT Takes the square root of the resulting value of the expression which follows the keyword.

example: .ASSIGN N = SQRT (A + E -C);

meaning: Assign the value of the square root of the sum of variables A and E minus variable C to variable N.

TBOOL — A built-in bit string (bit address 1-99) that can be used for flags that are automatically reset to zeros when ATG starts to process a new $TEST.

VAC — Assigns an rms value of ac voltage that should be used by ATG when evaluating the circuit between the indicated nodes.

example: .SET VAC (2, REF) = 0.1;

meaning: Apply the ac voltage of 0.1 rms between the nodes designated as node 2 and node REF.

VDC — The same as VAC except that it is concerned with dc levels.

5.5.4 System Variables

General

There are a number of keywords that are recognized as system variable names by ATG. These do not have to be defined by an ASSIGN statement prior to their use. They may be used in the same manner as any user-defined variable for the appropriate parameters. These variables can be divided into four groups according to their use. These groupings are:

Equivalent Circuit Parameters

B	X
CAP	Y
G	Z
R	

Characteristic Values of Device under test

BX	XX
GX	YX
RX	ZX

Test Parameters

DLY	HILIM	VBIAS
ERR	LOLIM	VS
FS	VALUE	

Node Designations

Number from 1 through 19
REF

These variable names are reserved keywords that may not be used for other names. The purpose of each system variable is presented in alphabetical order as follows:

B — Susceptance, calculated by ATG, between indicated nodes of the test configuration equivalent circuit, not including the device under test. Example: .NEED B[1, 2];

BX Susceptance, calculated by ATG of the device under test.

CAP Capacitance, calculated by ATG, between indicated nodes.
Example: .NEED CAP[1, 2];

D (Reserved for future use)

DLY Delay calculated by ATG for a circuit.

ERR Variable name where figure of merit may be saved.

FS Frequency Source to be used by ATG in evaluating the equivalent circuit.

G Conductance calculated by ATG between the indicated nodes of the test configuration equivalent circuit, not including the device under test.
Example: .NEED G[1, 2];

GX Conductance, calculated by ATG, of the device under test.

HILIM High Limit of component tolerance-actual value determined by adding the upper tolerance value to the nominal value of the component.

LOLIM Low Limit of component tolerance.

Q (Reserved for future use)

R Resistance, calculated by ATG, between indicated nodes of the test configuration equivalent circuit, not including the device under test.
Example: .NEED R[1, 2];

RX Resistance calculated by ATG of the device under test.

VALUE Nominal characteristic value of component to be tested.

VBIAS Name of a variable where a voltage may be saved.

VS Name of a variable where a voltage may be saved.

X Reactance, calculated by ATG, between indicated nodes of the test configuration equivalent circuit, not including the device under test.
Example: .NEED X[1, 2];

XX Reactance, calculated by ATG, of the device under test.

Y Admittance, calculated by ATG, between indicated nodes of the test configuration equivalent circuit, not including the device under test.
Example: .NEED Y[1, 2];

YX Admittance, calculated by ATG, of the device under test.

Z Impedance, calculated by ATG, between indicated nodes of the test configuration equivalent circuit, not including the device under test.
Example: .NEED Z[1, 2];

ZX Impedance, calculated by ATG, of the device under test.

5.6 TEST FORMAT INSTRUCTIONS

The FORMAT section of the library provides the means of translating the ATSL calculated test parameters into executable test language statements. The FORMAT section statements are Test Language statements with special format keys and parameter flags inserted in place of the actual value or instruction. Keys (keyspecs) are recognized by ATG as instructions to format information that is to be written into the test program or report file. Parameter flags (paramspecs) are recognized by ATG as instructions to insert data into the test program and report files. Test program file organization is also controlled by keyspecs. All keyspecs and paramspecs must be preceded by a percent (%) symbol which indicates to ATG that a keyspec or a paramspec follows. The following paragraphs describe the keyspecs and paramspecs:

5.6.1. Keyspecs

%-1A Instructs ATG to print page headers in the test program. This is used by default if no page header keyspec is included.

%1A Instruct ATG to skip the page headers in the test program. This keyspec remains in effect to the end of the current stream or until changed. When a new stream is entered, this keyspec must be specified or the default will be assumed.

%-2A Instructs ATG to print all numeric data with engineering suffixes, e.g., 1.6 K. This is used by default if no number format keyspec is included.

%2A Instructs ATG to print all numeric data with E suffixes, e.g., 1.6 E3. This keyspec remains in effect to the end of the current stream or until changed. When a new stream is entered, this keyspec must be specified or the default will be assumed.

%nA Instructs ATG to set the page length of the test program to the value of n, where n is a number larger than 20. The page length is set to 52 lines by default if this keyspec is not included. This keyspec remains in effect to the end of the current stream or until changed.

%-nA Instructs ATG to set the line length of the test program to the value of n, where n is a number from 21 to 80. The line length is set to 74 characters by default if this keyspec is not included. If n is 21 to 39, the %1A key spec must also be specified because the page header has a 40 character line.

%nB Instructs ATG to insert a new line (NL) and a tab in the test program line being output if the present character is within n characters of the line length specified by %nA.

%nF Instructs ATG to begin a new page before outputting the next line of the test program if the present line is within n lines of the page length specified by %nA. A value of 1 or more must be specified for n.

%nL There is a register which is automatically incremented by ATG each time a $FORMAT is traversed. It provides a means of generating labels which are unique within the test program for each component test procedure. This keyspec instructs ATG to replace the keyspec with the nth of these component unique labels.

%M Instructs ATG to print in the report file any messages resulting from ATG which are enabled by the comments level monitor option. These messages originate from .QUIT, .QUIT IGNORE, .TEST, and .SAY statements. They also originate from error messages in the .ERR file, from any ATG error messages, and from vetoes generated by ATG.

%1N Instructs ATG to replace the keyspec with the name of the component being tested. This is the name provided in the circuit description file. %N and %0N are recognized by ATG as this keyspec.

%2N Instructs ATG to replace the keyspec with a list of components which are connected in direct shunt between pins 1 and 2 of the component being tested. The keyspec is replaced with null, if there are no shunt components.

%3N ATG assigns a unique number for each component test procedure. This keyspec instructs ATG to replace the keyspec with this number for the current component. This number will be used by diagnostic routines, when applicable.

%4N Instructs ATG to replace the keyspec with the name of the current circuit board.

%5N Instructs ATG to replace the keyspec with the veto message followed by the component name. If there is no veto message for the current component, the keyspec is replaced with null. ATG generates veto messages internally, which are saved in the event of a veto.

%6N Instructs ATG to replace the keyspec with the name of the current test, e.g., RGRD.

%7N Instructs ATG to replace the keyspec with the highest unique component test procedure number that has been assigned by ATG.

%8N Instructs ATG to replace the keyspec with a label which consists of the name of the current component, as provided in the .CKT file, appended by the test number of the current device within the complex device, if applicable. ATG assigns test numbers for a complex component in the following way. The component is represented in the .WOR file as a primary component and by a series of subcomponents. It is assumed that there will be a test in the libraries for the primary component and one for each analog sub-component. The tests are numbered so that the first test is for the primary component and will not have a test number in the label. The exception to this is when the primary component is of type IC. In this case the first test will be numbered 1 to avoid a conflict with labels assigned in digital test procedures.
Example: The component is complex device RP9 which consists of prime devices R1 through R6. If the current test for R2 is the second test for the device RP9, the label inserted by ATG for this keyspec is RP9_1:.

%9N Instructs ATG to replace the keyspec with a label which consists of the current component whether it is a primary component in the .CKT file or a sub-component within a complex device. Using the same example of %8N, the label inserted by ATG for this keyspec is R2:.

%nP Instructs ATG to replace the keyspec with name of a nail that is assigned to the node indicated by "n." n is an integer from 1 through 20. ATG searches the circuit description file and inserts the name of the first nail found. The nodes are defined by .PIN statements and by the Guard chooser. More than one node may be specified in the keyspec up to a maximum of 9. In this case, ATG will replace the node list with a nail list that contains the first nail found on each node, e.g., %1,2P keyspec will be replaced with "6, 25," in the FORMAT statement if 25 is the first nail on node 1 and 6 is the first nail on node 2. The nail list is written in ascending order of nails, with a dash to represent a continuous series.

%-nP Similar to %nP except that ATG searches the circuit description for the second nail found on the node indicated by "n." If there is no second nail, the first nail will be used.

%20P Similar to %nP except that the node referenced is the REFerence node.

%-20P Similar to %-nP except that the node referenced is the REFerence node.

%nS Tells ATG the collating sequence in which the group of statements within the present set of braces ({ }) are to be included. When ATG assembles the finished test program, the test procedures that ATG generated will be inserted in ascending order of stream numbers. These stream numbers are "n" in the keyspec. There will be several groups of statements with the same stream number. ATG will arrange these within the stream in the order encountered. Each of these groups will have a number added to the stream number which denotes the order within the stream. The resulting number is called a page number in the test program. The form in which the test program is written is influenced by the %nA, %nB and %nF keyspecs. Each stream must have its own specification for these keyspecs or the defaults will be used. There is no carry-over from one stream to another.

%T Instructs ATG to replace the keyspec with the current component type.

%nV Instructs ATG to replace the keyspec with the current value of the parameter indicated by n, where n is a number from 1 through 8, defined as follows:

1. Upper tolerance limit resulting from the tolerance specified by a circuit description flagspec or by default. The value stored in the system variable HILIM.

2. Lower tolerance limit resulting from the tolerance specified by a circuit description flagspec or by default. The value is stored in the system variable LOLIM.

3. Nominal characteristic value of component as specified in the circuit description. The value is stored in the variable VALUE.

4. Voltage value stored in the variable VS by a .SET statement. Originally was used for source voltage; however, it may be used to store any voltage.

5. Frequency of source in capacitance and inductance tests specified by a circuit description flagspec (FREQ) or by default. The frequency is stored in the variable FS.

6. Voltage value stored in the variable VBIAS by a .SET statement. Originally used for bias voltage; however, it may be used to store any voltage.

7. Figure of merit value for a test configuration which is saved in the SELECT section. It is the value that is saved in the variable ERR to which a subsequent .SAVE statement argument will be compared in determining the "best yet" test configuration. Written in form nnnn.mm.

8. Number of component tests generated by ATG up to the present time. Written as a 4-digit integer. ATG automatically increments a register each time a test is written.

%% Instructs ATG to replace the keyspec with a %. This is necessary in order to have a % symbol included in the output statement. A single % symbol would tell ATG to look for a keyspec or paramspec that it would not find.

5.6.2 Paramspecs

The formats of a paramspec are:

1. %#specname#write-code

2. %#specname#write-code {character string}

where: %# Indicates to ATG that a paramspec follows

# Indicates to ATG that a write-code follows

specname 1. A format-variable-name or any flagspec in the circuit description which contains a message.

2. A specname that has special meaning to ATG, namely, NEWTST and NEWTYP. NEWTST is an instruction to ATG to insert into the program all the statements enclosed in the next set of braces ({ }) if this $TEST entry differs from the previous one in this stream or is itself the first to be used in this stream. NEWTYP is an instruction to ATG to insert into the program all the statements enclosed in the next set of braces if this is the first component of the current type to be evaluated.

write-code A code that ATG uses to make a decision about what should be used in replacing the paramspec. These decision codes are explained in the following paragraphs.

Write-codes

D — Used in the second paramspec format. Tells ATG to replace the current paramspec with the text string, that is included in the paramspec, if the specname is found. Otherwise, replace the specname with nothing.

F — Used in the first paramspec format. Tells ATG to replace the current paramspec in accordance with the following:

a) "specname" if the specname is found but has no value assigned.
b) "specname = floating-point-number" if the specname is found and has any numeric value assigned to it.
c) "specname = 'text-string'" if the specname is found and has a text string assigned to it.
d) nothing if specname is not found.

H — Used in the first paramspec format. Tells ATG to replace the current paramspec in accordance with the following:

a) nothing if specname is found but has no value assigned.
b) "floating-point-number" if the specname is found and has any numeric value assigned to it.
c) "text-string" if the specname is found and has a text string assigned to it.
d) nothing if specname is not found.

I — Used in the first paramspec format. Tells ATG to replace the current paramspec in accordance with the following:

a) nothing if specname is found but has no value assigned.
b) "decimal number without suffix" if specname is found and has a numeric value n, with $.001 < n < 1000$; otherwise, "floating-point number with suffix."
c) "text-string" if the specname is found and has a text string assigned to it.
d) nothing if the specname is not found.

N — Used in the second paramspec format. Tells ATG to replace the current paramspec with the text string, that is included in the paramspec, if the specname is not found. Otherwise, replace the specname with nothing.

5.7 SPECIAL CONSIDERATIONS IN LIBRARY DESIGN

The ATG process and the run-time test can become very time-consuming with some large circuit boards. The design of the library file can affect these times considerably. Therefore, this section discusses some of the techniques used by GenRad to minimize ATG time. It is important to the user who creates a User Library file to maintain these concepts.

5.7.1 Order of Configurations Within Tests

ATG processes configurations in a library procedure in the order in which they appear. To keep ATG processing time to a minimum, it is important to

arrange the configurations in the order in which they are most likely to be accepted. The guarded resistance test provides a good example of ordering configurations. It has been determined that the most likely configuration to be accepted by ATG is the one that uses the lowest source voltage, which has the source applied to the device pin that is connected to the circuit node that has the most connections, and which has the measure node guarded. The next most likely is with the source node swapped. Then the guard node is swapped. Next, the source polarity is changed. If a test is not yet accepted, this search process will be repeated at a higher voltage. The value of the resistor will determine the maximum source voltage that will tried. This is normally 5.0 volts for resistors over 1.0 megohm, 1.0 volt for resistors between 500 ohms and 1.0 megohm, and 0.2 volt for resistors under 500 ohms. These values can be changed by the user with the flagspecs RHIV and RLOV. Another means of controlling the order of ATG processing is with the flagspec FASTATG. This is used to speed up the ATG process by skipping the evaluation of seldom used configurations. The library procedure would contain a conditional jump in the procedures that would be skipped when the user specifies FASTATG. In the guarded resistance test, two compromises may result. There may be fewer 'superior' tests and, therefore, less test accuracy. There may also be a 'no test' condition because there are tantalum capacitors on the board and FASTATG skips negative source voltage configurations. Refer to Section 8 of this manual for a complete list of guarded resistance configurations.

5.7.2 Method of Inserting MUX Statements and Subroutines

Multiplexer functions are very time consuming at Run-time. Therefore, the library should be designed to have no redundancy in MUX statements. All testing that uses a particular MUX configuration should be done in consecutive test program "streams." As test procedures are written they are assigned to a particular stream. The streams are ordered so that one MUX statement can be inserted just ahead of a group of streams that use the same MUX configuration. Streams within a MUX group are further ordered to reduce changes in source and measure instrumentation parameters. For example, all the low source voltage tests are done first and then the next higher voltage tests are done, etc.

MUX statements are inserted in the appropriate streams after all component tests have been written. ATG searches the GBOOL bit string to determine if a particular MUX statement is required. During the writing of a test procedure the appropriate GBOOL bit is set to indicate to ATG that a test exists in the group of streams that requires the MUX statement.

Subroutines are inserted in a similar manner as MUX statements. Each time a procedure is written that uses a subroutine, the appropriate GBOOL bit is set. After all procedures have been written, ATG searches the GBOOL bit string to see which subroutines are required and inserts them at the end of the test program.

5.7.3 GenRad System and User Library Interaction

The user should be aware of which GBOOL bits are used by the system test library so that care can be taken not to disrupt the integrity of their intended use. Table 5-2 summarizes the contents and features of the GenRad System test library. Table 5-3 lists the GBOOL bits.

Table 5-2
TEST PROCEDURE CONTENT REFERENCE

TEST			xBOOL					COMPONENT TYPE(S), CDL SYMBOL	STREAM(S)
NAME	TYPE	DIRECTORY	G	Y	D	T	C		
OPENS	C								4
NOPEN	C								4
NOPSHRT	C								19
NOSHRT	C								20
FIXTURE	C								1
ISOPEN	G	NOCONT	46, 47			1–5		NO, OJ	4
NOTOPEN	G	CONT	24, 46			1–5		NC, J, F, L	4
2TR	G	RES	2, 20		1	3, 4		R, F, L	27, 29, 31
RGRD	G	RES	2, 20		1	2, 3, 4		R, F, L	26, 28, 30
RFORM	G	RES	2, 20			3, 4		R, F, L	32
RDIG	G	RES	40, 41					R	97
2TC	G	CAP	4, 20		1, 2			C	43, 45
CGRD	G	CAP	4, 20		1, 2	2		C	42, 44
CFORM	G	CAP	4, 20		2			C	46
2TCP	G	POL TANT	4, 20 4, 20		1, 2 1, 2			CP CP1	43, 45 43, 45
CPGRD	G	POL TANT	4, 20 4, 20		1, 2 1, 2	2 2		CP CP1	42, 44 42, 44
CPFORM	G	POL TANT	4, 20 4, 20		2 2			CP CP1	46 46
2TL	G	IND	4, 20		1			L	43, 45
LGRD	G	IND	4, 20		1	1, 2		L	42, 44
LFORM	G	IND	4, 20					L	46
POTFIX	G	POT	2, 34, 35					RV	27, 29
POTFIXG	G	POT	2, 34, 35					RV	26, 28
POTADJ	G	POT	2, 12, 32, 34					RV	11, 27, 29
POTADJG	G	POT	2, 12, 32, 34					RV	11, 26, 28
RVFIXG	G	POT	2, 34					RV	26, 28
RVADJG	G	POT	14, 20, 33					RV	15, 16
PFORM	G	POT	2, 12, 14, 20, 32, 33, 34, 35					RV	17, 33
JUNCTST	G	CRFORWD	8, 22		1			CR	65
CRFWD	G	CRFORWD	8, 22		1	1–9		CR	51
CRFWDG	G	CRFORWD	8, 22		1	1–9		CR	51
CRREVIG	G	LEAKAGE	8, 22		1	1		CR	61
CRREVI	G	LEAKAGE	8, 22		1	1		CR	61
CRFDFOR	G	CRFORWD	8, 22		1			CR	52
CRRIFOR	G	LEAKAGE	8, 22		1			CR	62
VZ	G	ZENER	10, 20			1		VZ	86
VZFORM	G	ZENER	10, 20			1		VZ	87
JFETFOR	G *	TRANCON	44, 45	1		1, 2, 10		NJFET, PJFET	79
SCRIM	G *	THYRIST	42, 45		1, 2	1–6		SCR	81
SCRVM	G *	THYRIST	43, 45		1, 2	2–6		SCR	83
SCRFORM	G *	THYRIST	43, 45		1, 2	1–4		SCR	84
QSIM	G	QGAIN	6, 26					QN, QP	71
Q	G	QGAIN	6, 26					QN, QP	71
QB	G	QGAIN	6, 26					QN, QP	71
QFORM	G	QGAIN	6, 26					QN, QP	72
RAIL	G *	OPAMP	45, 48		1, 2, 3	1		UK5	102
RAILFOR	G *	OPAMP	45, 48		1, 2, 3			UK5	102
TPDEC	U		44						1
TPMUX	U		2, 4, 6, 8, 10, 12 14, 20, 22, 40, 42, 43, 44, 46						1, 3, 10, 14, 24, 40, 50, 70, 78, 80, 82, 85, 95
TPEND	U								200, 240
TPSUB	U		8, 20, 22, 24, 26, 32, 33, 34, 35, 41, 45, 47						1, 250

Table 5-3
SYSTEM TEST LIBRARY GBOOL USAGE

GBOOL BIT	ITEM INSERTED	STREAM	TESTS USING ITEM
2	MUX statement for ohmmeter device (RM)	24	2TR, RGRD, RFORM, POTFIX, POTFIXG, POTADJ POTADJG, RVFIXG, all PFORMs
4	MUX statement for ACZ device	40	2TC, CGRD, CFORM, 2TCP, CPGRD, CPFORM, 2TL, LGRD, LFORM
6	MUX statement for Transistor Gain Tests	70	QSIM, Q, QB, QFORM
8	MUX statement and setup for Diode Tests	50	CRFWDG, CRFWD, CRFDFOR, CRREVI, CRREVIG, CRRIFOR, JUNCTST
10	MUX statement for Zener Diode Tests	85	VZ, VZFORM
12	MUX and set-up for adjusting Potentiometer	10	POTADJ, POTADJG, PFORM
14	MUX and set-up for adjusting variable resistors	14	RVADJG
20	Subroutine FAILSUB	1, 250	2TR, RGRD, RFORM, 2TC, CGRD, CFORM, 2TCP, CPGRD, CPFORM, 2TL, LGRD, LFORM, VZ, VZFORM
22	Subroutine FAILCR	1, 250	JCT, FWD, REV, FWDFORM, REVFORM
24	Subroutine FAILCONT	1, 250	NOTOPEN
26	Subroutine FAILQDCS Subroutine FAILXTOR	1, 250	QSIM, Q, QB, QFORM
32	Subroutine POTADJ Subroutine POTFAIL3	1, 250	POTADJ, POTADJG, PFORM
33	Subroutine VRADJ	1, 250	RVADJG, PFORM
34	Subroutine POTFAIL	1, 250	POTFIX, POTFIXG, POTADJ, POTADJG, RVFIXG, PFORM
35	Subroutine POTFAIL1	1, 250	POTFIX, POTFIXG, PFORM
40	ASSIGN LGC and MUX statements for all Resistors tested with digital stimulus	95	RDIG
41	Subroutine FAILPULL	1, 250	RDIG
42	MUX and set-up for SCR I measure technique	80	SCRIM
43	MUX and set-up for SCR V measure technique	82	SCRVM, SCRFORM
44	Form set-up for JFETs	78	JFETFOR
45	Subroutine FAILCOMP	1, 250	SCRIM, SCRVM, SCRFORM, JFETFOR, RAIL, RAILFOR
46	Set-up for all OPENS and SHORTS tests	3	NOTOPEN, ISOPEN
47	Subroutine FAILOPEN	1, 250	ISOPEN
48	MUX and Set-up for OpAmp RAIL tests	102	RAIL, RAILFOR

Digital Test Library Programming Guide – Section 6

6.1 INTRODUCTION

The purpose of the Digital Test Library (DTL) is to provide the Digital automatic Test Generator (DTG) with general test procedures for individual digital IC devices. As the test generator scans the circuit description file (.CKT) that describes a circuit board, it finds an entry for each digital IC on the board. This entry gives the IC's type number and lists how its pins are connected to nodes on the board. Using this information, DTG must write a digital test for the IC in digital test language (refer to the Test Programming manual, para 6.4). This it does by locating the IC type in the Digital Test Library; the entry for the IC contains a generalized procedure for testing the IC which, as far as possible, is not dependent on a particular circuit environment. Selecting and adapting those portions of the general library procedure that are appropriate, DTG then writes an IC test into the digital test program file (.DTP) that contributes the digital portion to the final, merged test program (.TPX or .TPG file, as shown in Figure 2-1).

Examination of a typical digital test program (.DTP) written by DTG will show it to consist of 2 main test sections. In the first section, all buses are tested, using BUSTEST blocks (as described in detail in the Test Programming manual, para 6.3). The second section contains a test for each individual digital IC on the board. Figure 6-1 illustrates how DTG produces the bus test section of the test program. Using IC connection information it finds in the circuit description (.CKT) file, DTG identifies all bused nodes and constructs a list of the ICs connected to each bus. It then consults the Digital Test Library to find how to place all of the ICs on the bus in a high impedance state, so that the test system can check the bus lines. Under the type name of each IC, DTG searches for a section called a disable procedure for the IC. As soon as DTG finds a disable procedure that can be used successfully on the circuit board, it copies the procedure from the library, allocates the required test nails from a list of available driver/sensors, and subsitutes the nail numbers for the IC's pin names. After making these changes, it writes the test steps into the test program.

For the second part of the bus test, called the BUSBUST section, DTG needs information on how to drive the outputs of each IC high or low. Again, it searches the Digital Test Library for the required procedures and writes those that can be used in the IC's circuit environment into the test program file.

Having written tests for the buses found on the board, DTG can proceed to write tests for the individual ICs. For each individual IC named in the .CKT file, DTG consults the library again, as shown in Figure 6-2. However, before a particular IC, or unit-under-test (UUT), can be exercised, DTG must take steps to isolate the UUT from its circuit environment. (This important aspect of in-circuit testing is explained in Test Set Preparation, para 1.3.) Therefore, using the .CKT file, DTG must first construct a list of all ICs directly connected to the UUT. Then it searches the library entries for these ICs, attempting to find disable (.DISAB) or inhibit (.INH) procedures that it can write into the program, so that the UUT can be tested with as little interference as possible from surrounding ICs. Finally, DTG searches the

library entry for the UUT to find a way of testing it -- a set of test steps that will exercise the IC's functions and indicate whether it is working properly or not. This part of the IC's library entry is called a .MAIN (alternatively, .SPECial) section. As with all other library procedures it has used, DTG evaluates the .MAIN section to find whether the test it prescribes can actually be performed -- that is, whether the IC's pins can be driven in the ways required, given the way the IC is connected to other components, signal lines, power supply lines, etc. If DTG finds that the .MAIN section's requirements can be met, it concludes the BURST section in the test program file by writing out the test statements given in the .MAIN library procedure.

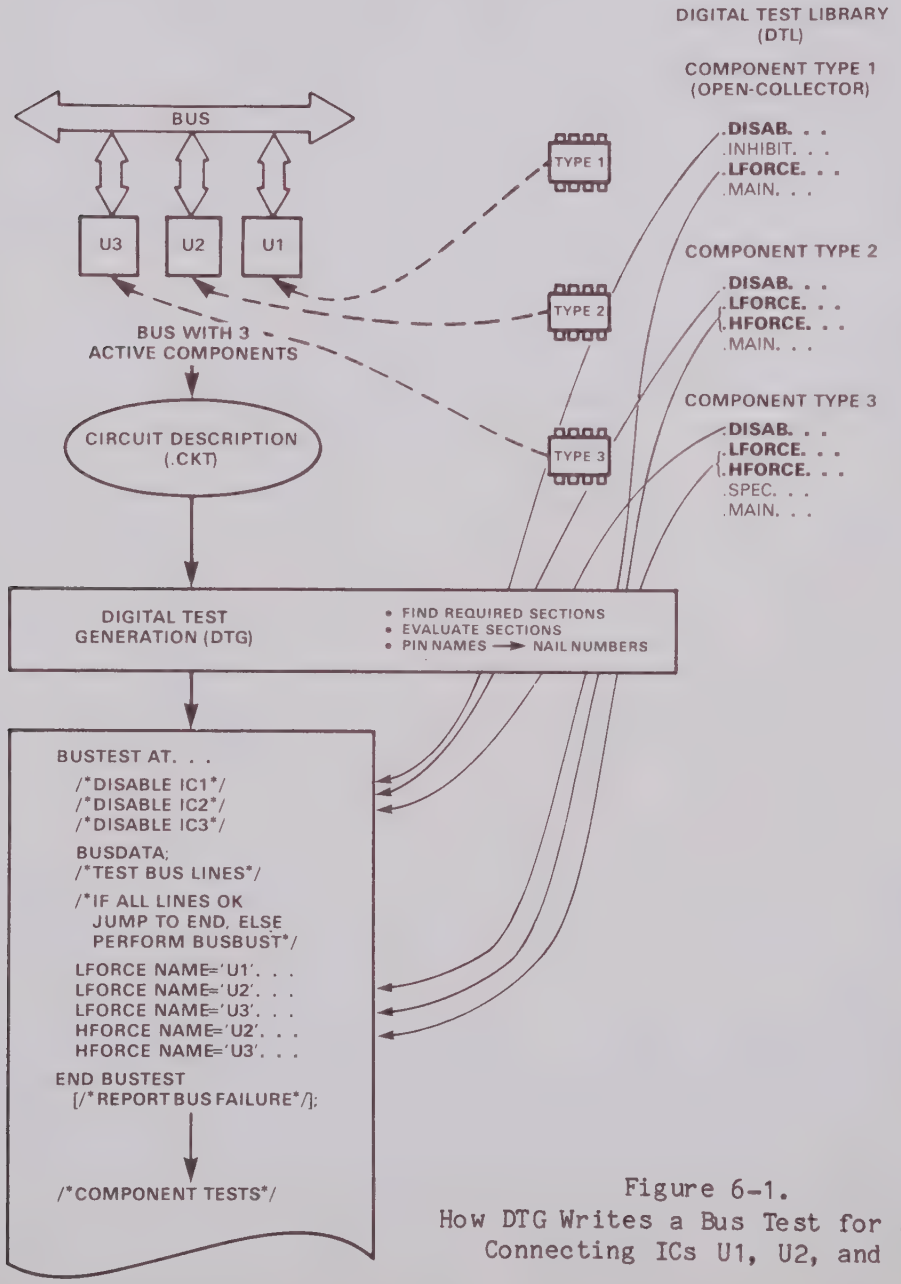


Figure 6-1.
How DTG Writes a Bus Test for a Bus Connecting ICs U1, U2, and U3.

Proceeding through the list of ICs described in the .CKT file, DTG writes a BURST section for each IC on the board, and it thus creates a complete in-circuit digital test for the board, one that checks all bus structures and all individual digital ICs. This test is, of course, limited by the amount and kinds of information available in the library; where a required procedure cannot be found, or the circuit configuration does not permit a procedure to be used, gaps will remain that the test programmer should later evaluate and possibly fill.

DTG's operation is actually more complex than outlined here; it proceeds in a number of steps and creates a number of intermediate files. Understanding the relationship between these files and processes can be helpful in understanding the function of the test libraries. Figure 6-3 shows these relationships and the flow of information during DTG.

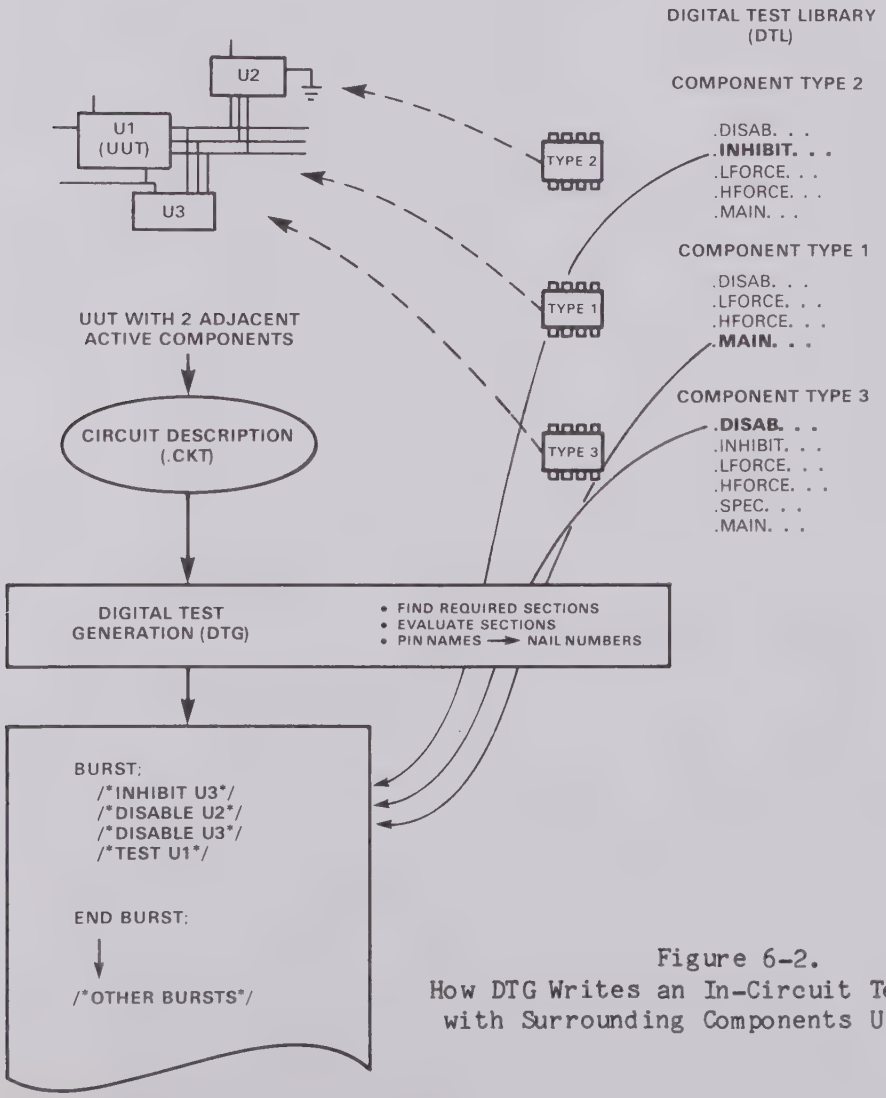


Figure 6-2.
How DTG Writes an In-Circuit Test for U1, with Surrounding Components U2 and U3.

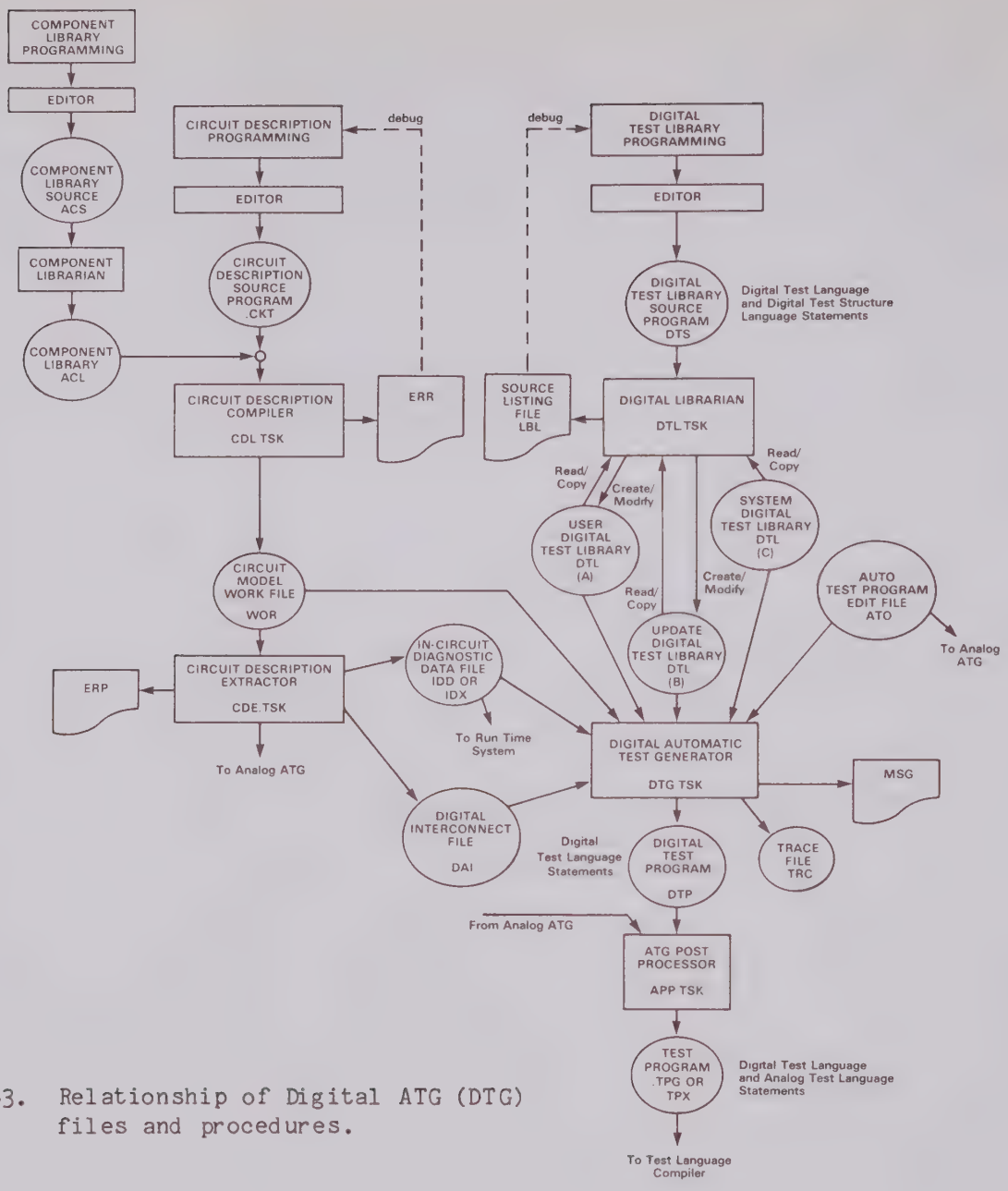


Figure 6-3. Relationship of Digital ATG (DTG) files and procedures.

Digital Test Libraries

The Digital Test Library (DTL) consists of 3 library files: the system, update and user libraries. The system library, which is part of the original system, may be copied by the user but not changed. The update library is changed each time updates are provided by GenRad. The user library is changed each time the user writes a new test or modifies a test that may have existed on any of the three DTL files. It is assumed that if the user has modified a test from the system library and placed it in the user library, the user test is the preferred version. Because the user library is searched first by DTG when looking for a procedure entry, user-created test procedures supersede all other procedures in the library. Updates supersede only procedures in the system library, which is searched last.

Libraries are stored in a file format that cannot be edited by the user. Changes and additions to the user and update libraries must therefore be made via the DTL monitor mode, which provides facilities for translating library procedures from the library format to text format, and vice versa. The user can enter a new library element, either by extracting and modifying an existing test from one of the library files or by creating an entirely new source file written in DTSL. The Digital Test Librarian then translates this source file into a library, or DTL, form that can be interpreted by the DTG. The librarian then inserts the test into the user or update test library so that it may be used during test generation. The user cannot make changes or insertions in the system libraries.

While a digital library procedure has the general form of an IC test written in the digital test language, it is not itself a test; it is rather a set of more general instructions that DTG can adapt to a particular circuit environment. (Refer to the Test Programming manual, Section 6, for a description of the test language.) A library procedure is written in a special, extended variant of the test language, called GenRad Digital Test Structure Language (DTSL). The test that DTG writes for an IC is in the form of a BURST or a series of BURSTs that can be inserted in a board test program. Hence, DTSL provides constructs similar to those in the test language for setting and testing pin states, but, unlike the test language, it specifies a test in terms of device pin names rather than fixture nail numbers. (Features of the test language not used within BURSTs, e.g., floating point arithmetic, read and write operations, are not reflected in DTSL.) In addition, there are statements that allow a library procedure to specify pin states conditionally, for example, to specify that in a group of 5 IC pins at least 1 must be set high and that DTG can choose any 1 of the 5. Using general instructions of this type, DTG can create test language statements for a given IC which work successfully in many different circuit environments.

Other DTSL constructs allow conditional insertions or choices among alternative sections of the procedure, so that different test strategies can be provided for different circuit environments and test situations. Programmers familiar with a macro-assembly language will recognize in DTSL many of the functions used in conditional macro assembly. An IC description in the circuit file (.CKT) acts somewhat like a macro call, passing to the Digital Test Library the IC's name and its pin connections. After finding an entry for the IC name, DTG generates a test procedure by 'expanding' the library call, evaluating the pin connections in the circuit description and choosing those portions of the library procedure that will work effectively in the IC's actual circuit environment. DTG also uses the IC's connection data to identify other components that might interfere with the test. From the library entries for these other components, it extracts test statements that disable their outputs or otherwise inhibit them from affecting the test. Where bus structures can be identified, DTG writes a test for the bus by extracting bus-related information about each IC on the bus from the libraries. Table 6-1 lists the keywords recognized in DTSL.

Table 6-1
DTSL KEYWORDS

$	.NAND.	CDD	INDEP	PB
.AND.	.NOR.	CLEAR	IS	PD
.CHOOSE	.NOT	CONN	IST	PH
.CLOCK	.OR.	CYSNC	ITT	PHASELOCK
.CLUST	.OUTPUT	DEF	KNOWN	PL
.CONTEXT	.PERIOD	DST	LET	PN
.DECLARE	.QUIT	END	LGC	PU
.DH	.REPEAT	EXITIF	LO	RETURN
.DISAB	.SAY	F	LOOP	SET
.DISJ	.SIZE	FAIL	LOOPIF	SETIND
.DL	.SPEC	FAILS	LOSLEW	SIGNAL
.DU	.SUB	FAST	MAXTIME	SUBROUTINE
.ELSE	.TRI	FLAG	NINC	SYNC
.ELSEIF	.TRIGGER	FLAGFAIL	NOPHASELOCK	T
.END	.UNDRIV	FLOOP	NOSYNC	TEST
.EXITIF	.XOR.	GND	NP	TG
.FI		GOSUB	OFF	TH
.FREE		GOTO	OG	TINC
.FREQ	ACTIVE	GROUP	OH	TL
.FTEST	AT	HI	OI	TO
.GOSUB	BRANCH	HISLEW	OL	UG
.HEAD	BREAK	HOLD	OM	UNDEF
.HFORCE	BSTRING	I	ON	USING
.IF	BURST	IC	OPENC	VCC
.INH	C	ID	OPENE	VCDH
.INPUT	CALL	IG	OS	VCDL
.INSERT	CDA	IH	OU	VCST
.LFORCE	CDB	II	OV	VTT
.LOOP	CDC	IL	PASSES	
.MAIN				

6.2 WRITING A DIGITAL LIBRARY PROCEDURE

A library procedure for a digital device must provide DTG with information about how to verify that the device is working properly, as well as information about how to prevent the device from interfering with tests of other digital devices and buses. For a general discussion of the theory of digital testing, refer to the Test Set Preparation manual, Section 1.

Before a programmer starts to write a Digital Test Library Source (DTS) procedure for an IC device, he or she must first learn the functions of that device. For some MSI and LSI devices, this learning process may be a major portion of the time spent in developing a procedure. The programmer will then start sketching out the test steps needed to exercise all the pins and functions of the device. Once this is done, the programmer considers all the ways in which the IC can be connected in a circuit. This consideration will usually lead to a number of alternative test strategies. The primary purpose of DTG is to choose among these alternatives. Therefore, additional language statements must often be added to the basic test procedure statements to provide DTG with the information it needs in the decision process. These additional structuring statements do not themselves produce test program statements but instruct DTG how to choose or modify the test statements in the procedure, based on the device's circuit connections. DTG can thus use a single library procedure to

write many different tests, each unique to the device's particular circuit environment. When a new procedure is needed, a test programmer will add a new model to a user DTL, using the Library Management procedures described in Section 3.

6.2.1 Incremental Context of Pin States

The Digital Test Structure Language is described in detail in para 6.3; many of its commands are similar to those in the test language. Its use of pin state context is similar to nail context in the test language, but there are some important ramifications that deserve attention here, before we examine DTSL programming techniques.

While the digital test language requires the programmer to specify test nail states, DTSL works primarily with pin states, and each step in a test consists of a list of the states for all pins on the IC. DTG later translates these into nail states, so that the states of each pin become the states of the test nail assigned to it in the circuit description (.CKT) file. The state of a pin can be described, of course, as high or low; in addition, however, DTSL can describe a pin as being sensed (if it is an output), as driven (if an input), or as irrelevant to the test. The command IH(2), for example, causes (input) pin number 2 on an IC to be driven high; ID(2) disconnects the pin from the driver. Both the condition of being driven high and the condition of being disconnected can be regarded as states of pin 2.

Only the pin states that change from one test step to the next need be mentioned in a DTSL command. During test generation, DTG keeps a continuing record, called the context, of the current states of all the pins of an IC under test and writes out each TEST LGC step in terms of the entire pin context, that is, for all pins on the IC. This means that at every TEST LGC command all active pins are tested -- driven or sensed -- and only those pins are disregarded whose states have been set to 'disconnect' or 'ignore.' Once pin states are changed by a command, the new states are the context until explicitly changed again. Consider the test statement

```
IH(1,2,3) OI(4) OL(5,6);
```

This means that pins 1, 2, and 3 of an IC are to be driven high, pin 4 is not sensed, and pins 5 and 6 are checked for low outputs. The statement

```
IL(2);
```

then generates a test step that is actually equivalent to

```
IL(2) IH(1,3) OI(4) OL(5,6);
```

The last 3 commands belong to the implicit pin context of the test. The pin context is a record of all pin states that have been defined in the course of the test; at the beginning of a test procedure all pins are automatically set to OI (output ignore) or ID (input disconnect). To save space, DTG does not write out nail states that already belong to a test step's context.

The pin context does not necessarily, however, extend through all portions of a procedure. At the beginning of a SUBroutine, for example, the context is automatically set to OI and ID for all pins, as at the beginning of a procedure and, after the SUBroutine finishes, the context that existed before the call is restored. This means that the pin context immediately following a

SUBroutine call is not affected by the call, and the context within the SUBroutine is always the same, regardless of the context of the test step where it is called. In other words, the pin context of a SUBroutine is entirely local; it does not interact with the context of the calling routine or program.

Conditional sections of a procedure are sections that may or may not be used in generating a given test for the device. (Examples are the CLUSTER and IF-THEN-ELSE constructs.) At the end of a conditional section DTG restores the pin states that exist immediately before the section, but at the entry to the section DTG leaves the context unaffected. This means that the conditional section can make use of the context passed to it from the statement preceding it, and yet the unconditional sections will always execute in the same way, regardless of how conditional sections are used. Another way of saying this is that the pin context is carried into a conditional section, but not out of it.

6.2.2 Sample DTG Test Transformations

As mentioned in para 6.1, DTSL statements bear a strong resemblance to statements in the digital test language. Because the writer of a library procedure cannot know in advance how an IC will be connected, the library procedure must be formulated to take account of all possible circuit environments (unless it is known that the IC will always be connected in a particular way). This means that the test should include all input and output pins and cover all reasonable uses of the IC. When DTG evaluates the procedure, it compares each test step to the circuit description and checks to see whether the step can be executed. If, for example, an output pin is not connected to the circuit, it obviously cannot be tested, and those portions of the procedure that test it will be discarded. Where pins are wired high or low, DTG adds their states permanently to the pin context. If a step in the procedure specifies a different state, it is discarded.

The following example will give an impression of how DTG adapts a simple, general procedure for a 3-input NAND-gate to the requirements of the circuit described in Figure 6-4. The procedure consists largely of statements that function like test language statements. In addition, a special DTSL construct is introduced as an example of the extensions to the test language found in DTSL. Examples of other, more complex techniques that DTG can use for evaluating a procedure will be provided later.

The test for a 3-input NAND-gate consists, of course, of nothing more than its truth table. The Digital Test Library includes the following test procedure (sometimes called a model) for the 7412 (some sections are omitted here). Since the IC contains 3 identical functional units, it is convenient to write the generalized test for just one of these units. This can be done with a special DTSL construct called an equivalent-set-name (explained in more detail in para 6.3). This construct allows us to break the IC into 3 equivalent units and write a single test for all 3. In the INPUT statement, the sets A, B, and C are defined as the first, second, and third input to each NAND-gate. The open-collector outputs are designated as the set Y. When it writes a test for the IC, DTG will proceed through these lists, writing first a test for input pins 1, 2, and 13, with output pin 12. Then it will write a test for the gate with input pins 3, 4, and 5, with output pin 6, and so on. The equivalent set names are defined in a section of the procedure called the HEADer that describes the IC:

```
###     CDL FILE: SC:7412.CKT,  COMPILED 30-AUG-82
###
     1 :%CIRCUIT;
     2 :U1  7412
                1=W1,2=W2, 3=W3,4=W4,5=GND,6=W6,
                7=GND, 8=W8,9=VCC,10=VCC,11=VCC,
                12=W12,13=W13, 14=VCC;
     3 :
     4 :%ADAPTER: 2270;
     5 :H101     =W1;
     6 :H102     =W2;
     7 :H103     =W3;
     8 :H104     =W4;
     9 :H105     =GND;
    10 :H106     =W6;
    11 :H107     =GND;
    12 :H108     =W8;
    13 :H109     =VCC;
    14 :H110     =VCC;
    15 :H111     =VCC;
    16 :H112     =W12;
    17 :H113     =W13;
    18 :H114     =VCC;
```

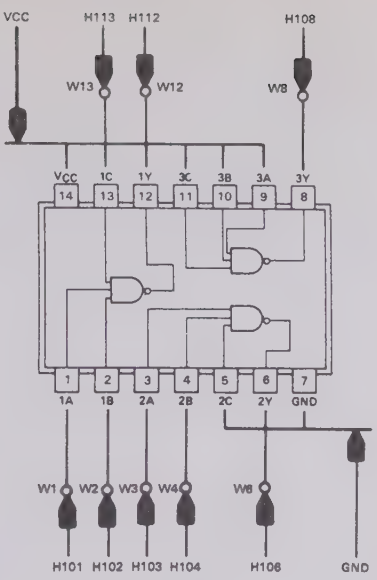


Figure 6-4. Circuit Description for Triple NAND-Gate.

```
/*7412 TRIPLE 3 NAND OPEN-C*/
.HEAD;
        .SIZE 14;
        .INPUT (1 3 9=A,2 4 10=B,13 5 11=C);
        .DH (A,B,C);
        .OPENC (12 6 8=Y);
.END HEAD;
```

Because the IC contains open-collector outputs, the tester must connect a pull-up resistor in the first step of the test. This is done with the SET LOGIC statement, $PU(Y) in line 3 below; with each use of the model test, DTG substitutes the current output pin for Y. The inputs must, of course, be connected to logic drivers (with an IC statement). IH(A,B,C) then drives all 3 gate inputs high. OS(Y) connects a logic sensor to the output and OL(Y) specifies that a low is to be expected on that output. The output can only be low if all inputs are high; the rest of the test steps check that the output goes high and stays high as each input pin is driven low. (The statement OH(Y) -- expect a high at the output -- remains in effect in statements 6, 7, and 8 because it now belongs to the context, as was explained in para 6.2.1.)

```
1 :     .MAIN;
2 :     BURST IST=10U;
3 :     $PU(Y);
4 :     IC(A,B,C) IH(A,B,C) OS(Y) OL(Y);
5 :     IL(A) PH(B,C) OH(Y);
6 :     IL(B) PH(A,C);
7 :     IL(C) PH(A,B);
8 :     ID(#) OI(#);
9 :     $PN(Y);
10:     .END MAIN;
```

Let us now examine how DTG creates the BURST required to test the 3 gates in this IC when it is connected as shown in Figure 6-4. During DTG processing, the equivalent-set variables (e.g., A, B, C, Y) are replaced with the test nail numbers connected to the pins in the circuit description. DTG encounters no

difficulties in writing a full test for the first NAND-gate, and thus it produces a full, 4-step test, shown below. The inital comment (written by DTG) shows how the ADAPTOR section of the circuit maps the IC pins to test nails. The test then prescribes how the test nails will be driven or sensed at each step.

```
/** TEST PROGRAM FOR IC U1 (7412)
PIN NAIL COMMENT
  1 101
  2 102
 12 112
 13 113
 **/
        $ PU(112);
        IC(101,102,113) IH(101,102,113) OS(112) OL(112);
        IL(101) OH(112);
        IL(102) IH(101);
        IL(113) IH(102);
        ID(101,102,113) OI(112);
        $ PN(112);
```

For the second NAND-gate, however, we actually need a different test, since the input pin 5 is tied low. If the test above were used, this gate would automatically fail the first TEST LOGIC step, where the output is expected low. Discovering that it cannot drive pin 5 high, as required in the step 4 of the model test, DTG compresses the step. It shortens the IC (input connect) command, retaining only the pins that it can control; it eliminates IH(A,B,C) because it cannot drive all 3 pins high, as well as the 'output low,' OL(Y), that is supposed to result, and proceeds to step 5. However, it uses as much of step 4 as it can carry out to establish the pin context (that is, it connects the available inputs and the output pin), and it adds a low on pin 5 (the pin that is tied low) to the pin context. From line 5 of the test model, the model's second TEST LOGIC statement, it generates the first test step shown below. DTG again eliminates commands that require pin 5 (nail 105) to be driven high.

```
/** TEST PROGRAM FOR IC U1 (7412)
PIN NAIL COMMENT
  3 103
  4 104
  5 105 LOW
  6 106
 **/
        $ PU(106);
        IC(103,104) IL(103) IH(104) OS(106) OH(106);
        IL(104) IH(103);
        IH(104);
        ID(103,104) OI(106);
        $ PN(106);
```

The third NAND-gate has all 3 of its inputs, pins 9, 10, and 11, tied high, which further restricts DTG's ability to write a full test for the gate. In fact, all it can do is check that the output pin (pin 8) is low when power is applied to the IC.

```
/** TEST PROGRAM FOR IC U1 (7412)
PIN NAIL COMMENT
  8 108
  9 109 HI
 10 109 HI
 11 109 HI
 **/
        $ PU(108);
        OS(108) OL(108);
        OI(108);
        $ PN(108);
        END BURST; /* U1 */
```

6.2.3 Trace Mode with Digital ATG

A record of ATG's actions in accepting, rejecting, and modifying sections of a library procedure can be produced by selecting the FULL trace option on the ATG monitor page during digital test generation. The digital trace can be most useful in seeing how an especially complex test has been derived from the library. Analog test generation should be disabled, since the analog trace information is usually voluminous and not relevant for digital test library development. Furthermore, the test should be edited to eliminate sections you do not wish to examine. The digital trace file can become very large. Choose ATG options as follows.

```
Analog Test Gen [...  NO]

Digital Test Gen [YES ...]

Enable Trace Mode [... FULL].
```

If ATG is run from a Batch mode file (.BCH, created by typing the command SAVER on the ATG page), a Batch Listing (.BLI) file will be created containing a detailed record of ATG's actions. As shown below, it indicates which INHIBits, DISABles, and test sections have been chosen. Some excerpts from the .BLI file for the triple NAND-gate test developed in para 6.2.2 appear below. These trace statements represent an intermediate stage in the creation of the test program (.TPG) file; they show how DTG has evaluated the model test before it substitutes test nail numbers.

For the second NAND-gate in Figure 6-4, the trace shows the following evaluation, indicating that it has added the first model statement to the context, but that it has rejected the test statement.

```
IC(A,B,C) IH(A,B,C) OS(Y) OL(Y);
                                        /*TEST LGC rejected*/
IL(A) PH(B,C) OH(Y);
IL(B) PH(A,C);
IL(C) PH(A,B);
ID(#) OI(#);
```

For the third gate, it shows that all the test statements except the first were rejected.

```
IC(A,B,C) IH(A,B,C) OS(Y) OL(Y);
IL(A) PH(B,C) OH(Y);
                                        /*TEST LGC rejected*/
IL(B) PH(A,C);
                                        /*TEST LGC rejected*/
IL(C) PH(A,B);
                                        /*TEST LGC rejected*/
ID(#) OI(#);
```

6.2.4 Guidelines for Developing a Procedure

Developing a new library test procedure often encompasses 2 somewhat separate phases -- finding an appropriate electrical interface to the tester, and finding an appropriate test strategy. Once written, the procedure must be verified by running it with a known-good device.

Development Hardware

For developing new library procedures it is desirable to have available a test fixture containing a number of zero-insertion-force sockets of various sizes. Alternatively, a test fixture with an edge connector for small circuit boards can be prepared, as shown in Figure 6-5. The socket pins should be wired to similarly numbered test nail connections (e.g., pin 1 to nail 1 or 101). The test nails can be directly wire-wrapped to the socket pins or connected through DIP switches that allow simulation of opens or shorts. Using this fixture, the user can debug the library procedure by writing a short, quickly compiled test program for the socket board, rather than a program for the full circuit board on which the IC is used. The file containing the procedure will be listed as the DTLusr option on the ATG monitor page during this development phase.

Special modeling cards are available to 2272 and 2275 users. On the 2272 this card plugs directly into a connector on the Operational Verification Fixture. On the 2275, the card plugs into the Customer Training and Modeling Fixture. Refer to Section 4 of the Test Set Preparation manual for details of these modeling cards and fixtures.

As will become clear in the sample test procedure in Figure 6-7, a library procedure usually consists of at least a HEADer section describing the device, a DECLARE section that defines variables used in the procedure, a DISABle or INHibit section that tells DTG how to prevent the device from interfering with the testing of other, adjacent devices, and a MAIN section containing the

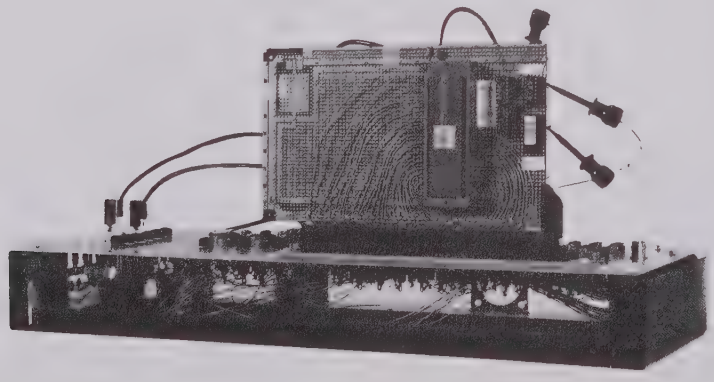


Figure 6-5. Fixture for Developing IC Test Procedures.

statements that make up the actual test. In addition, some tests may incorporate other DTSL sections. INSERTs will be used where there are repeated sections of code that can be written once as macro-instruction definitions, then inserted in the MAIN section where required. A SPECIAL section will be used when there is a simpler, alternative test procedure that can be used when certain conditions are met. A SPECIAL section will replace the MAIN section when DTG recognizes that its conditions are met.

The order in which the sections typically appear is as follows:

.HEAD	(Device description section)
.DECLARE	(Variable and device pin declarations)
.CLCOK	(Defines SpeedPlus timing parameters)
.DISAB	(Method of disabling the device)
.INH	(Method of inhibiting device outputs)
.INSERT	(Macro procedures called by later sections)
.HFORCE	(First part of bus test method)
.LFORCE	(Second part of bus test method)
.SPECial	(Alternative test method)
.MAIN	(Test procedure begins here)

Digital library procedures can become quite complex, so the writer of a new procedure should try as far as possible to alter procedures for devices already in the library rather than devise completely new tests. (Use the DTL monitor page to extract procedures from the library files.)

Declarations

The first step in developing a library procedure is to write HEADer and DECLARE sections. (The precise syntax of these and all other sections is described in para 6.3. Examples are shown in para 6.2.5 and 6.2.7.) A HEADer section is required to describe the basic electrical features of the device. These include the number of pins on the IC, and a listing of the pins that are inputs, outputs, tri-state, or inactive. In addition symbolic names (such as RESET or CE) can be assigned to the pins in the HEADer to make the procedure more readable. GenRad library procedures are usually written in terms of symbolic pin names.

If a device can (or must) be tested using the optional SpeedPlus digital testing subsystem, a special .FTEST section of the header lists what resources the component requires from the subsystem. These requirements typically include lists of device pins that will require special timing (clock and trigger) nails, pins that require SpeedPlus driver/sensor nails, and a range of acceptable operating frequencies. A separate .CLOCK statement can also declare the required clock parameters. Clock parameters can, however, also be inserted into the digital test program (.DTP) file after digital test generation via an .ATO file (refer to Test Set Preparation manual, para 5.2). This method is especially useful where a circuit board imposes specific timing requirements (e.g., synchronization to a particular on-board clock frequency) not met by the more general timing parameters given in a library procedure. A clock frequency can also be specified in the VALUE section of the circuit description (.CKT) file (refer to Test Set Preparation manual, para 6.7). Other overall test parameters, such as clock and trigger logic levels, are assigned default values, which can be changed by editing the .DTP file. Where digital or analog tests must be performed at SpeedPlus timing nails, DTG automatically inserts appropriate DECLARE NAILBIND statements into the .DTP file (refer to Test Programming, para 6.2, DECLARE NAILBIND); no nail-binding declarations are used in the library procedures.

Bused or associated pins, such as data or address pins, should be identified with DECLARE GROUP statements. A GROUP is a powerful DTSL construct that allows an ordered collection of pins to be treated as a single arithmetic variable. For example, if 16 address pins are grouped together under the variable name ADDR, they can all then be set high with the statement IG(ADDR=X'FFFF'). An address can be incremented simply by writing ADDR = ADDR + 1. Where arithmetic variables are required for performing pin state calculations, these can be declared as bit strings (BSTRING) and manipulated in similar ways, except that DTG will not concern itself with whether corresponding pins are actually present.

DISABles and INHibits

Following the header and declare sections, it is necessary to include information that allows DTG to test other ICs on the same bus or electrically adjacent to the IC being tested. This is done with one or more DISABle and INHibit sections. The DISABles will be used to disconnect the IC's outputs from buses or lines that it shares with other ICs. The INHibits will be used to prevent the IC from changing its output state during a test of another IC.

Usually a bused IC will have one or more tri-state or device enable pins that can be used to set its output pins to a high-impedance state, effectively disconnecting them. Sometimes other disable methods are available. However it is implemented, the DISABle section needs to tell DTG how to disable the outputs without affecting the outputs or operation of other devices. Often a component can be disabled effectively by activating its reset pin. However, reset lines are typically connected to a number of devices on the bus and would impair the operation of those devices as well; hence, DTG will reject a disable that uses a reset pin if it requires the reset line for the UUT. For this reason, other disable methods should be written as well. Many devices provide more than one control pin for disabling, and a DISABle section should be provided for each one separately.

DTG will insert only the first usable DISABle that it finds, so the most preferable disabling methods should be placed first. When 2 or more pins are available, it is important that the disable section chosen gain control of every pin that is not tied high or low, since any uncontrolled pin may pass 'glitches' from test activity occurring elsewhere on the board. (Refer to the Test Set Preparation manual, para 1.3.4, for a discussion of error mechanisms in digital testing.) Thus, the first DISABle section should be one that drives all available disabling pins. A DISABle or INHibit section is considered usable only if every step it contains can be executed.

Next come INHibit sections that give DTG additional information on how to prevent the IC from interfering with the testing of other circuits. If the IC cannot be disabled (if it has no chip enable or tri-state lines, or if these are tied to power or ground), circuit conditions such as feedback loops and free-running clock signals can cause the IC to toggle its outputs. This may interfere with the testing of other circuits and give spurious test results. The INHibit procedures tell DTG what it can do to keep the IC in a stable condition or how the IC can otherwise be prevented from interfering with other circuits. With TTL devices it is easier for the tester to drive logic highs than lows, so where an INHibit section cannot set an output to a high-impedance state, it should drive it high.

Test systems equipped with the SpeedPlus option can make use of DISABle and INHibit sections that execute on the SpeedPlus hardware. The statements

.DISABle USING and .INHibit USING declare that the section may contain FAST blocks and list the clock and nail requirements for the SpeedPlus subsystem. Statements within the section that are to execute via the SpeedPlus hardware must appear between the keywords FAST and END FAST. The test system will assign component pins to the required SpeedPlus test nails and provide timing specified by an associated .CLOCK statement in the library procedure.

Because only a limited number of SpeedPlus test nails are available on any test system, SpeedPlus DISABles and INHibits should be included in a procedure only when they allow significantly better tests than can be obtained using the standard test hardware. They should usually be placed after the standard DISABle and INHibit sections, so that workable standard sections will be chosen in preference to the SpeedPlus sections; in this way the SpeedPlus nails will be conserved for the test procedures that need them most.

Ahead of every FAST block created by .INHibit USING and .DISABle USING must appear a SET CLOCK statement in the .DTP file. This SET CLOCK establishes the timing parameters that will be used by the SpeedPlus hardware during the FAST blocks. DTG will therefore attempt to extract a clock statement from the library procedure and write it into the test program, ahead of the burst in which the INHibit or DISABle procedure is used. If the component being tested and all other inibited or disabled components use only standard test statements (without FAST blocks), this clock statement will be used exclusively for controlling a single, high-speed disabled or inhibited component. However, if the component being tested also requires the SpeedPlus clock (i.e., it contains one or more FAST blocks of statements), a conflict may arise -- that is, the disabled component may require one set of clock parameters, and the component under test may require another. Only one SET CLOCK statement can be in effect at a time, and DTG will write only one clock into the .DTP file. DTG gives precedence to the clock required by the component under test. Consequently, DTG will reject any DISABle or INHibit procedure whose clock requirements are incompatible with the requirements of the component being tested. If DTG can find no compatible DISABle or INHibit procedure in the libraries, it places a note in the .MSG file, and the user must then provide appropriate modifications. This is most conveniently done by adding a set of statements to the .ATO file associated with the test program. (For example, clock waveforms required by the disabled device can be assigned to clock driver channels not being used by the FAST portions of the component test.) Clearly, the same type of conflict can also arise between 2 or more disabled or inhibited components that require signals from the clock module.

DTG regards the clock requirements of a device it must disable or inhibit as compatible with the requirements of the UUT only if both use the unnamed, default clock specified with .FREQ statements that have overlapping clock ranges. For 2 such .CLOCK statements, DTG writes a single SET CLOCK into the .DTP file, using the lowest frequency allowed by both statements.

Hence, as much as possible, library procedures for the SpeedPlus subsystem should rely on the default clock parameters rather than specifying detailed, special timing parameters. Frequency ranges should be specified for as wide a range as appears usable with a component, and the low end of the range should represent the lowest frequency at which correct operation can be expected. If special clock waveforms or other parameters are required, a detailed clock statement can be substituted after automatic test generation by means of an ATO file.

At this point, with HEADer, DISABle, and INHibit sections in place, almost enough information is available for DTG to process the procedure. If a dummy MAIN section is added, consisting of nothing but the statements,

```
.MAIN;
   SET LGC;
.END MAIN;
```

ATG will be able to process a description of a circuit containing the component, although it will not produce a test for the component. This can be useful when the most pressing need is to get a working skeleton test for a board, rather than a working library element.

Circuit description files produced by the CKTGEN program may from time to time reference incomplete 'skeleton' models in the digital user library containing a dummy .MAIN, .END MAIN block. CKTGEN adds such a procedure when the user describes the circuit connections of an IC type that CKTGEN cannot find in the libaries. The skeleton procedure allows DTG to produce a rudimentary working test program from a .CKT file before a full library procedure is developed for the new IC type. CTKGEN's skeleton test, however, contains no DISABle, INHibit, or force sections, and these must be added by the user, along with an appropriate MAIN section. A skeleton procedure is identified in the library by the keyword .SKELETON, and DTG marks the use of a skeleton as an error, although it does not reject the procedure. Refer to the Test Set Preparation manual, para 6.5, for more information.

Inserts and Subroutines

The parts of the procedure written so far allow DTG to prepare tests for other devices on a circuit board, but nothing has been provided to test the IC itself. Now some thought must be given to how the IC functions and to the kinds of test statements that will be needed to exercise it. It will often prove convenient to define a number of basic test operations, such as clocking the device or shifting a bit string, that will be used repeatedly in the test. DTSL allows the programmer to do this by writing modules that can be inserted into the general flow of the procedure at compile time via an INSERT call. The INSERT module is put ahead of the main procedure in which it is used and is identified by a leading statement that contains the keyword INSERT and the name which the user assigns to the module.

DTG does not identify INSERTs when it expands them, and they are therefore difficult to identify in the test program (.TPG) unless marked by identifying comments (which DTG will retain). Longer inserts should both begin and end with comments.

Unlike an INSERT, a DTSL SUBroutine does not produce new code in the BURST when it is called, so it can be used to keep a BURST small. However, its usefulness is limited by the fact that pin states are not passed into the routine; a SUBroutine always starts with all inputs disconnected and all outputs ignored. Because INSERTs simply add test steps to the in-line flow of the program, the pin state context passes into them and out. When a commonly used routine must make use of the pin states existing at the point where it is called, without affecting the context of the calling routine, it should be written as an INSERT that consists of a CLUSTer section. The context will be passed to the INSERT, but any context changes that occur within the CLUSTer will not be passed back to the calling routine or main program.

Bus Test Section

Because they can use the INSERTs that have been devised for the device test, the bus test sections are often not written until the INSERTs are available.

When a bus fault is detected, a test is usually performed to determine which IC on the bus is causing the fault. To generate the test, DTG must be told how it can force all outputs of each IC on the bus high and low, and how it can disconnect the IC from the bus. Disconnection is provided for by the DISABle section; instructions for forcing the outputs must be provided in HFORCE and LFORCE sections.

Each HFORCE and LFORCE sections contains a sequence of test statements that will result in all bused outputs being set to high or low states. Because an open-collector output cannot drive the bus high, an HFORCE instruction cannot be entered for it; for the same reason, an LFORCE cannot be included for an open-emitter output.

On systems equipped with the SpeedPlus option, the force sections, like the INHibit and DISABle sections, can be written to execute at higher test speeds. The statements .LFORCE USING and .HFORCE USING will allocate SpeedPlus resources for the bus test and permit FAST blocks in the bus test, using clock parameters indicated or implied by the USING construct. If a conflict arises with the clock required to test the bus, DTG will reject the required force section, placing a note in the .MSG file. As with a rejected INHibit or DISABle, compromise clock and force statements must then be added to the .DTP file via an editor. Again, SpeedPlus force sections should be included in a procedure only when they are genuinely necessary, and they should usually be placed after the standard force sections.

MAIN Section

Now that a skeleton for the test procedure has been created, the test itself can be designed. At this point an important decision about the ultimate use of the test needs to be made. If the IC is always used in the same or similar circuit environments, or if it is unlikely to be used on other circuit boards, there is no point in writing a generalized procedure. A DTSL test can be written in a form very similar to a test language BURST.

If the IC is likely to appear in many dissimilar circuit contexts, however, it is worth the extra effort to write a larger and more general procedure that DTG can adapt to the various circuit boards on which the IC might appear. This kind of procedure provides for many possible circuit environments by extensive use of CLUSTer, CHOOSE, or IF statements, which allow DTG to evaluate and select various segments of the procedure. If, for example, an IC contains sections which may be unused in some applications, the tests for these sections can be written as CLUSTers. If a circuit description then indicates that these sections are not connected, DTG will ignore the segments contained in the CLUSTers. Likewise, there may be a number of ways to enable a device, depending on how it is used. Each of these ways can be defined as an INSERT and, when the device must be enabled, a CHOOSE I statement will cause DTG to search the list of enable methods for one that works in the IC's circuit environment. Using the IF-THEN-ELSE construct, the programmer can specify explicitly how the IC's circuit connections should be evaluated for choosing segments of the procedure. A number of DTSL functions are provided for evaluating pin contexts; they return true-false values that can provide the basis for DTG's use or rejection of segments in the procedure.

To write the MAIN section, the programmer must devise a test strategy for the device. The first goal of the test should be to make sure that each functional pin can be set high and low, so that correct connection to the circuit board nodes can be verified. Then the device's functions should be tested. For smaller ICs, the test can be constructed from the device's truth table; larger devices should be tested by verifying functional units within the IC.

Every MAIN section should end with the statement

```
ID(#) OI(#);
```

so that all drivers and sensors will be disconnected before the burst terminates. If omitted, DTG will insert this statement at the end of the burst.

6.2.5 Sample RAM Test Procedure

The process of writing a typical digital library test procedure is illustrated in the following paragraphs. The 2111A 256x4 RAM shown in Figure 6-6 is used as the example. Figure 6-7 shows the complete library procedure developed in these paragraphs. This and the example in para 6.2.7 show typical uses of DTSL statements; for a detailed description of DTSL, refer to para 6.3.

Declarations

The first programming step is to define the IC. This is done by writing a Header section for the procedure. The Header for the 2111A would be:

```
.HEAD;
   .SIZE 18;
   .INPUT(1=A3,2=A2,3=A1,4=A0,5=A5,6=A6,7=A7,
          9=OD,10=CE2,15=CE1,16=RW,17=A4);
   .TRI(11=IO1,12=IO2,13=IO3,14=IO4);
.END HEAD;
```

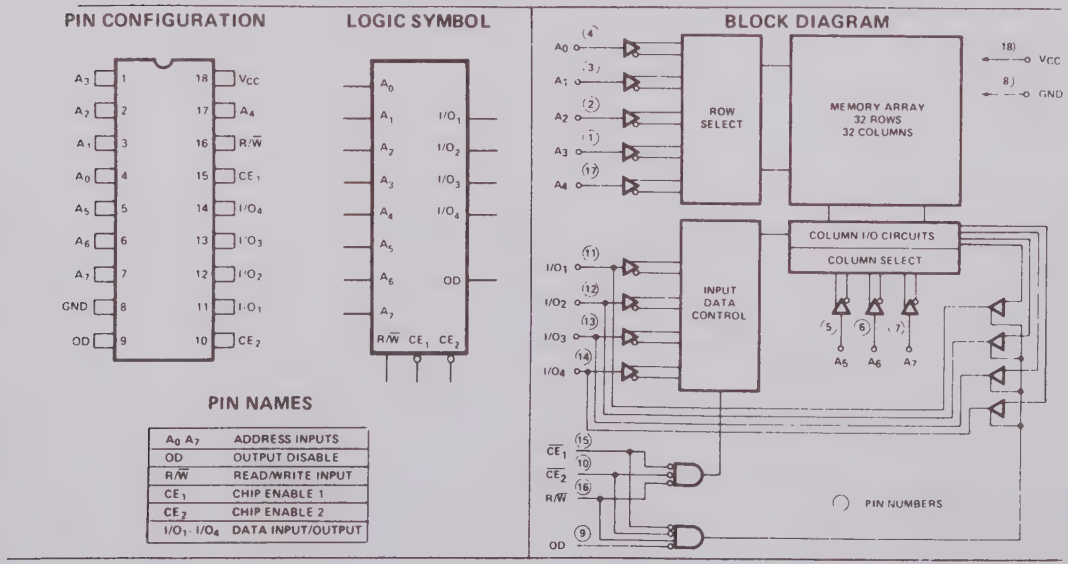


Figure 6-6. Design Data for the 2111A 256x4 RAM.

```
/*2111A          RAM TEST 256 X 4 STATIC*/
.HEAD;
        .SIZE 18;
        .INPUT (1=A3,2=A2,3=A1,4=A0,5=A5,6=A6,7=A7,9=OD,10=CE2,
                15=CE1,16=RW,17=A4);
        .TRI (11=I1,12=I2,13=I3,14=I4);
.END HEAD;

DECLARE GROUP ADDR(A7,A6,A5,A4,A3,A2,A1,A0);
DECLARE GROUP DATA(I4,I3,I2,I1);
DECLARE BSTRING ZERO(8);
DECLARE BSTRING ONES(8);
DECLARE BSTRING NEXTADDR(8);

.DISAB(DATA);
        IH(CE1);
        .HOLD(CE1);
.END DISAB;

.DISAB(DATA);
        IH(CE2);
        .HOLD(CE2);
.END DISAB;

.DISAB(DATA);
        IH(OD);
        .HOLD(OD);
.END DISAB;

.INH (DATA);
        IL(RW);.HOLD(RW);
.END INH;

.LFORCE(DATA);
  IH(OD) IL(CE1,CE2,A7,A6,A5,A4,A3,A2,A1,A0,I4,I3,I2,I1) IH(RW);
  IL(RW);
  IH(RW);

  ID(I4,I3,I2,I1) IL(OD);
.END LFORCE;

.HFORCE(DATA);
  IH(OD) IL(CE1,CE2,A7,A6,A5,A4,A3,A2,A1,A0) IH(I4,I3,I2,I1,RW);
  IL(RW);
  IH(RW);

  ID(I4,I3,I2,I1) IL(OD);
.END HFORCE;

.INSERT WRITE=2;
  OI(DATA) IC(DATA) IH(OD) IG(ADDR=?1,DATA=?2);
  IL(RW);
  IH(RW);
.END INSERT;

.INSERT READ=2;
  ID(DATA) OS(DATA) OG(DATA=?2) IL(OD) IG(ADDR=?1);
.END INSERT;

.MAIN;
LET ZERO=B'00000000';
LET ONES=B'11111111';
LET NEXTADDR=B'00000001';
IC(ADDR,CE1,CE2,RW,DATA,OD) IL(DATA,ADDR) IH(CE1,CE2,RW,OD);
IL(CE1,CE2);
/*WRITE ALL ONE'S TO LOCATION ZERO AND READ IT BACK*/
.'WRITE'(ZERO,ONES);
.'READ'(ZERO,ONES);
/*WRITE ZERO'S TO LOCATION ZERO AND READ IT BACK*/
.'WRITE'(ZERO,ZERO);
.'READ'(ZERO,ZERO);
/*WRITE LOCATION N AND READ LOCATION N THEN READ LOCATION ZERO*/
.REPEAT D'8';
        .'WRITE'(NEXTADDR,ONES);
        .'READ'(NEXTADDR,ONES);
        LET NEXTADDR=NEXTADDR*D'2';
        .'READ'(ZERO,ZERO);
.END REPEAT;
ID(#) OI(#);
.END MAIN;
```

Figure 6-7. Complete Library Test Procedure for the 2111A 256x4 RAM.

The beginning and ending of the section must be indicated by the statements .HEAD; and .END HEAD;. The number of pins on the IC is specified by the .SIZE 18; statement. Each input pin is identified and assigned a functional name. The name is not required but is used to keep track of pins by their functions. Next, the output pins are identified and assigned functional names. The outputs of the RAM are bus connections that are tri-state devices. Thus, these pins are identified by using the .TRI(); statement.

The next step illustrates the use of GROUPs, a powerful DTSL construct that allows groups of pins to be manipulated as single entities and treated as arithmetic variables (defined in para 6.3). For a memory IC it is useful to declare the device's address inputs as one group and the outputs as another. Descriptive group names may be chosen, such as ADDR and DATA. These group statements are:

```
.DECLARE GROUP ADDR (A7,A6,A5,A4,A3,A2,A1,A0);
.DECLARE GROUP DATA (I4,I3,I2,I1);
```

In the testing of a RAM there are many pin state calculations associated with reading and writing data to the address bus and the I/O bus. These calculations can easily be accomplished by declaring bit string variables, which can be manipulated as numerical quantities in the procedure. These manipulations are similar to GROUP operations, except that DTG will not check for missing pins. Bit strings must be defined prior to the section in which they are used, so it is common practice to put all declarations together just after the HEADER section. The following bit strings are used in this procedure.

```
.DECLARE BSTRING ZERO(8);
.DECLARE BSTRING ONES(8);
.DECLARE BSTRING NEXTADDR(8);
```

The DECLARE BSTRING statement assigns a name to the string and the number of bits in the string.

Disables and Inhibits

The 2111A has 2 chip enable pins that can be used to disable it, as well as an Output Disable pin, so 3 DISABle sections are provided.

```
.DISAB(DATA);          .DISAB(DATA);          and .DISAB(DATA);
     IH(CE2);               IH(CE1);                  IH(OD);
     .HOLD(CE2);            .HOLD(CE1);               .HOLD(OD);
.END DISAB;            .END DISAB;               .END DISAB;
```

The DISABle sections start with the section keyword DISAB and (in parentheses) a list or group name of the output pins that are to be disabled. These are followed by statements that set input pin states that are required to disable the bused outputs. The last command indicates the input pins that must be held in the disabling state during the test of an adjacent IC or bus. DTG will choose the first disable method that can be implemented.

The 2111A RAM can be inhibited in the following INHibit section, which holds it in the Write (to memory) state. In this state the device will place no signals on the bus.

```
.INH(DATA);
   IL(RW);
   .HOLD(RW);
.END INH;
```

For the 2111A, the INHibit and DISABle sections are very similar. In both cases the output pins in the group called DATA are being isolated from the circuit. In the DISABle sections the chip enable CE1 or CE2 is the input that must be set and held high for isolation. The read/write (RW) input pin must be set and held low for isolation in the INHibit section. However, other ICs will be inhibited in other ways, for example, by holding one or more AND-gate inputs low, or by holding OR-gate inputs high.

Bus Test Section

If a fault is discovered on a data bus to which a 2111A is connected, ATG must be shown how to set all the 2111A data pins high or low (in the read-from-memory state), as described in the explanation of bus testing in the Test Set Preparation manual, para 1.3.5. (The address pins are inputs only. Since they cannot drive the address bus, they cannot interfere with the operation of other ICs on the address bus.) First the IC must be enabled by resetting OD and pulling the 2 CE pins low. The same IL instruction is used to place lows on the data lines and the address 00 on the address bus. To clock in the data word, RW is pulled low; bringing it high again reads the word out to the data bus. The bus is then disconnected (ID) and the IC is disabled by pulling OD low.

```
.LFORCE(DATA);      /* LFORCE */
IH(OD) IL(CE1,CE2,A7,A6,A5,A4,A3,A2,A1,A0,I4,I3,I2,I1) IH(RW);
IL(RW);
IH(RW);
ID(I4,I3,I2,I1) IL(OD);
.END LFORCE;
```

The HFORCE section functions in the same way, except that highs are written to and read from the device.

Initialization Section

The bit strings that were named in DECLARE statements must be initialized to known values prior to use in the burst. This is done in the burst by using LET statements:

```
LET ZERO=B'00000000';
LET ONES=B'11111111';
LET NEXTADDR=B'00000001';
```

These statements use the respective binary numbers to set all 8 bits in the ZERO string to 0, to set all 8 bits in the ONES string to 1, and to set the least significant bit in the NEXTADDR string to 1 and the remainder to 0. (Note that this syntax is different from that of the test language.)

Inserts

Two operations are basic to everything a RAM device does: reading and writing to the data bus. Each of these operations involves 2 sets of pin states, one for the address and one for the data to be transferred.

Accordingly, 2 INSERT modules are provided that each take an address and a data parameter. One is the write-to-memory procedure and the other the read-from-memory procedure. These are as follows:

```
.INSERT WRITE=2;
/* INSERT WRITE */
     OI(DATA) IC(DATA) IH(OD) IG(ADDR=?1,DATA=?2);
     IL(RW);
     IH(RW);
.END INSERT;

.INSERT READ=2;
/* INSERT READ */
     ID(DATA) OS(DATA) OG(DATA=?2) IL(OD) IG(ADDR=?1);
.END INSERT;
```

The names of these two INSERTs are WRITE and READ. The =2 after each name indicates that two arguments must be defined, for use in the routine, when the INSERT is called by the main procedure. To write to the device, the tester must ignore the DATA lines as outputs [OI(DATA)] and connect them as inputs [IC(DATA)]. It must then bring the Output Disable high [IH(OD)]. The ?1 and ?2 in the OG and IG statements indicate where the INSERT arguments will be used to place data on the data and address pins.

It is advisable to place comments at the beginning of each INSERT, to identify code that is generated from the insert. A comment should also identify the end of a longer INSERT. DTG does not itself identify INSERTs when it expands them.

Main Section

Having described the IC, declared pin groups, and provided various functional modules, the programmer now writes the main procedure for testing the IC, which starts with the statement .MAIN; and ends with .END MAIN;. This is the section that implements the test strategy; for a RAM device an adequate test can consist of writing both 1s and 0s to all locations, then verifying that they can be read back. The MAIN test procedure statements are listed as follows, with explanations of each type of item as it first occurs:

```
.MAIN;
```

This DTSL structuring statement is not passed to the generated program. DTG will insert an identifying label and a BURST statement at this point, unless the MAIN section already begins with a BURST statement. If DUT is called U2 in the circuit description, the statement U2:BURST; will appear in the test program.

First, the test hardware must be connected and the device initialized.

```
IC(ADDR,CE1,CE2,RW,DATA,OD) IL(DATA,ADDR) IH(CE1,CE2,RW,OD);
IL(CE1,CE2);
```

These statements say to connect all input pins to drivers, to drive the inputs DATA, ADDR, CE1, and CE2 low, and then to drive inputs RW and OD high. Assuming, for simplicity, that the IC pins are connected to test nails with the same numbers as the pins, the test language statements generated from these 2 statements will be:

```
IC(1-7, 9-17)
  IL(1-7,10-15,17)
  IH(9,10,15,16);
IL(10,15);
```

All comments inserted in a library procedure will be carried over into test programs where the procedure is used if "program comments" option is selected on option page. It is always useful to explain the operation of the procedure with descriptive comments. The comment line

```
/* WRITE ALL ONES TO LOCATION ZERO AND READ IT BACK*/
```

describes the purpose of the next two statements.

```
.'WRITE' (ZERO,ONES);
.'READ' (ZERO,ONES);
```

These two statements tell DTG to insert the procedures named WRITE and READ into this location in the program. The arguments used in the WRITE and READ routines are ZERO and ONES. These arguments were defined earlier in the program as bit string names and were set to particular values by a LET statement. The value of ZERO is inserted into the WRITE and READ routines where the ?1 appears because ZERO was the first argument listed in the calling statements. The value of ONES is inserted where ?2 appears.

The comment line

```
/* WRITE ZEROS TO LOCATION ZERO AND READ IT BACK*/
```

describes the function of the next two commands.

```
.'WRITE' (ZERO,ZERO);
.'READ' (ZERO,ZERO);
```

These two statements use the same two inserts as before but with a different second argument. The previous use of these inserts writes zeros to the ADDR group and ones to the DATA group and then checks the reading of ones from the DATA group. The current use of these inserts is the same except that the DATA group is written with zeros and read for zeros.

The next series of commands will step through a similar procedure for each bit in the address word and data location of the memory. The memory is reset and sensed for zeros after each new location. This is noted by the next comment.

```
/* WRITE LOCATION N AND READ LOCATION N THEN READ LOCATION ZERO */
```

Since the procedure to check memory locations is a repetition of 4 statements with the incrementing of the address, these 4 statements can be included in a REPEAT construct.

```
.REPEAT D'8';
```

The DTSL REPEAT construct does not produce a LOOP in the test program. Instead, this statement tells DTG to insert, 8 times, the statements found from here to the END REPEAT statement. The first 2 statements in the REPEAT loop again use the WRITE and READ routines to check the writing and reading of ones to each memory address bit.

```
.'WRITE' (NEXTADDR,ONES);
.'READ' (NEXTADDR,ONES);
```

The address is incremented by

```
.LET NEXTADDR=NEXTADDR * D'2';
```

which multiplies the existing address (or NEXTADDR group) by 2. This in effect moves the 1 from the least significant bit of the starting NEXTADDR value (00000001) through each of the 8 bits in the NEXTADDR group. The LET statement is thus executed by DTG and is used to alter the 'READ' and 'WRITE' statements each of the 8 times they are inserted. No bitstring calculations are performed at runtime.

```
.'READ' (ZERO,ZERO);
```

ensures that writing into other addresses does not cause writing into address zero. In other words, it ensures that other addresses do not short to zero.

```
.END REPEAT;
```

closes the REPEAT loop. When all these locations have been checked, the test is complete and the final command statement

```
ID(#)  OI(#);
```

disconnects all inputs and ignores all outputs.

Note that this is not the most thorough test conceivable. Clearly there is a compromise that must be made between speed of execution and thoroughness, but by toggling all inputs and outputs high and low and exercising most of the IC's decoding logic, the IC's essential functions have been tested far more quickly than could be done with an exhaustive test. If any of the address lines fails to be decoded, the word ONES will write to address 0 and 'READ'(ZERO,ZERO) will fail. Nevertheless, only a sampling of addresses has been checked, and only 1s have been read from those addresses. Users who find they need a more exhaustive test can, of course, easily add the required steps.

The end of the MAIN group is

```
.END MAIN;
```

DTG replaces this library statement with END BURST;.

A procedure intended for use on systems having the SpeedPlus option can include 1 MAIN section that uses the SpeedPlus hardware. Like the SpeedPlus DISABle and INHibit sections, this main section follows the syntax,

```
.MAIN USING (<clock>, <SpeedPlus test pins>, <SpeedPlus timing pins>);
  <SpeedPlus test statements>
.END MAIN;
```

As with other sections, DTG selects the first MAIN section that it can successfully implement. Because a standard MAIN cannot be made dependent on any pin conditions, it is always used when DTG encounters it. For this reason, a .MAIN USING section must precede the standard MAIN section. If DTG finds the clock statement and SpeedPlus nails the .MAIN USING section requires, it will

write a FAST test for the component; otherwise it will proceed to the standard test (if one is available).

6.2.6 VLSI Test Techniques

As the complexity of an IC increases, so does the difficulty of writing a completely comprehensive test. Full functional verification of a microprocessor, for example, could require from 10,000 to millions of test vectors (input/output bit patterns). Hence, it is usually necessary to find a method of testing such devices that does not blindly attempt to verify all possible combinations of inputs and outputs. Rather than regarding the device as a 'black box,' the test designer should try to make some probable assumptions about the device's inner workings and design a series of tests that exercise the IC's internal functional units individually. This approach can produce a highly effective test without the difficulties that would result from writing the kind of exhaustive truth-table test used for smaller devices.

The first and most important part of developing a VLSI test procedure is to acquire a thorough understanding of the device's logical and electrical functioning. Large-scale peripheral devices, like PIOs and UARTs, often require special attention to their initialization requirements.

Microprocessors and similar devices also require careful initialization; indeed, a major part of the effort in developing a microprocessor test will be devoted to learning how to 'bring up' the processor on the circuit board. In addition, the processor instruction cycles must be synchronized to the tester. In many cases this can be done simply by assigning a test nail to the processor's clock pin and synchronously stepping the processor through a series of instructions. In some cases, however, multi-phase clock inputs or clock rates higher than the 227X can supply will be required.

As a first step, then, the writer of a VLSI test should determine that the device is electrically testable by studying the manufacturer's data sheets and examining test procedures for similar devices. The standard 227X hardware allows only a restricted view of the UUT, set by the mininum IST (drive-to-sense) time of 600 ns and the minimum ITT (clock) period of 3.6 us. These specifications imply that events occurring less than 600 ns after an input stimulus can be seen by the sensor electronics only if they persist until the sensor strobe occurs. Burst stimuli cannot be applied at a rate above about 280 kHz. This restriction will affect mainly the testing of dynamic devices that require rapid refresh of internal registers. (Refer to the discussion of the 8085 test in para 6.2.7 for methods of overcoming this limitation.) The electrical requirements of an IC's inputs and outputs can pose problems. The 227X driver will typically slew at around 30 V/us into 100 ohms resistive, and at about 10 V/us into nodes that are being back-driven, a rate that is too slow for some IC inputs. A test nail typically presents a 100-to-300 pF load to an IC output pin, which some devices will not be able to drive properly. In addition, the electrical path between a driver/sensor and the test nail may distort stimulus waveforms in ways that cause particularly sensitive inputs (e.g., clock pins) to function unreliably.

If timing, loading, or slew rate requirements cannot be met by the 227X driver/sensors, special hardware may be required in the test fixture. This may be as simple as a TTL buffer to clean up a clock waveform (as required for the 6502 test) or as complex as the set of outboard timing circuits, relays, and bus transceivers sometimes required for the 8086. Use the ATO feature to disable fixture clocks (and other clocks) when testing other components on the board (rather than modifying each version of the test program by hand).

Alternatively, systems equipped with the SpeedPlus digital testing subsystem can use tests that take advantage of the subsystem's faster test rates and special timing capabilities. Because every 227X system has only a limited number of SpeedPlus test and timing nails, library procedures should be written to run on standard test nails whenever possible. (Because the SpeedPlus test nails can be programmed in the same way as standard test nails, standard component tests can be performed at all nails in the system.) In many cases, a component can be tested adequately within the restrictions imposed by the standard digital hardware, but the SpeedPlus hardware will perform quicker or more effective tests. When a library procedure contains both kinds of test, DTG will write the SpeedPlus test into the test program if the procedure's hardware requirements can be met; otherwise it will select the less efficient standard test. Finally, a few components can be tested effectively only at high test rates or under very stringent test timing parameters. The library procedures for these devices will contain tests meant to run only on the SpeedPlus subsystem. Such devices should always be listed first in the circuit description (.CKT) file, so that DTG will allocate SpeedPlus resources for them before it provides for components that could also be tested with the standard hardware. Refer to Test Programming, para 4.1.3, for a description of the SpeedPlus subsystem programming capabilities. Many programming differences from the standard hardware can be grasped from the detailed descriptions of the SET CLOCK and TEST CLOCK statements in para 6.3 of the same manual.

Manufacturer's design specifications should not always be accepted as test specifications. Many processors with dynamic registers can run considerably slower at room temperature than their specified minimum clock frequency, which usually applies to the maximum operating temperature. Likewise, rigid timing or slew rates may be specified to ensure proper system rather than device operation, so that an adequate functional test of the device by itself can easily tolerate the 227X timing limitations. Functional testing should not be used to verify component timing parameters.

The initial phase of the test that initializes a device is typically the most tedious to develop. While much of the device initialization can often be accomplished by clocking a reset line in a specified way, other parts of the device may require individual initialization. Experimentation may be required to obtain reliable reset operation. For this purpose, a special fixture with zero-insertion-force connectors of various sizes can be useful (see Figure 6-5).

Once an adequate electrical interface to the tester has been provided for a VLSI device, a test procedure must be conceived. If possible, a synchronous test should be designed that uses a 227X driver nail to clock the device. Provision must, of course, be made to disable the on-board clock. Alternatively, an on-board clock can be enabled or a clock in the fixture can be provided, and a single-step test can be designed that uses a processor WAIT or HALT pin to simulate synchronous operation. (Examples appear in para 6.2.7.)

Since it is impractical to test all possible combinations of inputs and outputs, a VLSI test should try to accomplish the following objectives.

Toggle all pins high and low
Exercise all major functional blocks
Verify all addressing modes
Verify all data types (e.g., byte, word)
Access all internal registers
Perform typical user instruction sequences

Occasionally, test patterns for exhaustive testing of a VLSI device are available from its manufacturer, but more often a test must be designed without exact knowledge of the device's internal architecture. Guides for designing a thorough test can be found in the block diagram (usually provided as part of the device documentation) and in the structure of the device commands.

Often a functional or addressing mode will correspond directly to a bit in the instruction, so that a set of representative test instructions can be chosen by having each mode bit set and cleared at least once. The device should, however, be further analyzed to determine that each functional unit indicated on the block diagram (e.g., ALU, registers, counters, decoders) and each data path (e.g., from an accumulator to a memory address latch) is tested. Interrupts should be tested and status flags verified. If possible, timing should be examined by strobing sensor inputs at appropriate intervals.

Systems equipped with the SpeedPlus testing option have additional hardware for high-speed digital testing. If a test that can otherwise be performed on the standard digital test hardware requires a clock having a frequency higher than 280 kHz (the maximum test rate of the standard hardware), the SpeedPlus clock module can be synchronized to the test program to provide one or more continuous clock signals to the UUT at frequencies up to 20 MHz. The test fixture hardware required in some test procedures (such as the 8085 test described in para 6.2.7) can often be simulated by the SpeedPlus subsystem by using the SpeedPlus clock signals and short blocks of FAST test statements. Tests that demand complex timing or high test rates can make use of the SpeedPlus high-speed driver/sensor nails and triggering facility. Refer to the Test Programming manual, para 4.4, for a description of the SpeedPlus subsystem. Special DTSL constructs for generating SpeedPlus tests are described in para 6.2.9.

6.2.7 Developing a Microprocessor Test

The development of a test according to the guidelines of para 6.2.6 is illustrated with examples from both simple and relatively complex microprocessor tests. Some illustrations are from the relatively simple test for the R6502 [1]. The second test, given in full, is from the Digital Test Library for the more complex I8085 [2]. The 8085 is one of the more difficult microprocessors to test, and the test procedure therefore illustrates some techniques useful for hard-to-test devices. However, because it is taken from the system library, this test represents a more general approach than might be needed for a one-time-only application.

R6502 Initialization

As mentioned in para 6.2.6, it is generally preferable to write a synchronous test for a processor, using a 227X driver to clock the device while other driver/sensors drive and check the device's inputs and outputs. This kind of test allows the DUT to be synchronized to the tester easily. Because the 6502 can be run with a clock rate as low as 25 kHz, no timing difficulties will arise (except that a TTL buffer must be provided in the test fixture to obtain the required 10 ns rise and fall times for the clock line). The example of Figure 6-8, taken from the test for the R6502, shows how one can reset the 6502 and begin to test for proper instruction execution.

[1] Trademark of Rockwell International, Box 3669, Anaheim, CA.
[2] Trademark of Intel Corp., 3065 Bowers, Santa Clara, CA.

```
/*MCS6502                    MOS TECHNOLOGY CPU*/
/* 18-JUN-82 */
.HEAD;
        .SIZE 40;
        .INPUT(2=RDY,4=IRQ,6=NMI,37=CLKO,38=SO,40=RSET);
        .OUTPUT(3=CLK1,7=SYNC,9,10,11,12,13,14,15,16,17,18,19,20,
                22,23,24,25,34=RW,39=CLK2);
        .TRI(26,27,28,29,30,31,32,33);
        .DH(RDY,IRQ,NMI,SO);
.END HEAD;

.DECLARE GROUP ADDR(25,24,23,22,20,19,18,17,16,15,14,13,12,11,10,9);
.DECLARE GROUP DATA(26,27,28,29,30,31,32,33);


.INH (DATA,ADDR,RW,SYNC);
        IL(RDY);
        LOOP=D'10' IL(CLKO);
        IH(CLKO) END LOOP;
        .HOLD(RDY);
.END INH;


.MAIN;
        .SAY"REFER TO APPLICATION NOTES FOR TESTING MCS6502 CPU.";
        IC(RDY,IRQ,NMI,CLKO,SO,RSET,DATA) PH(RDY,NMI,SO) IH(IRQ,CLKO)
        IL(RSET) NP(DATA);
/*POWER UP CYCLE*/
        LOOP=D'10';
        IL(CLKO);IH(CLKO);
        END LOOP;
        IL(CLKO);IH(RSET,CLKO) OS(RW) OH(RW);
        LOOP=D'6';
        IL(CLKO);IH(CLKO);
        END LOOP;
        IL(CLKO) OS(ADDR,CLK1,CLK2) OG(ADDR=X'FFFC') OH(CLK1) OL(CLK2);
        IH(CLKO) IG(DATA=X'00') OL(CLK1) OH(CLK2);
        IL(CLKO) OG(ADDR=X'FFFD') OH(CLK1) OL(CLK2);
        IH(CLKO) IG(DATA=X'04') OL(CLK1) OH(CLK2);
/*INSTRUCTION FETCH;CLEAR CARRY FLAG(DONE FOR*/
/*INITIALIZATION PURPOSES)*/
        IL(CLKO) OG(ADDR=X'0400') OS(SYNC) OH(SYNC,CLK1) OL(CLK2);
        IH(CLKO) IG(DATA=X'18') OL(CLK1) OH(CLK2);
        IL(CLKO) OG(ADDR=X'0401') OL(SYNC,CLK2) OH(CLK1);
        IH(CLKO) OL(CLK1) OH(CLK2);
/*INSTRUCTION FETCH;LOAD X INDEX W/MEMORY(ABS.ADDR.MODE)*/
/*AND STORE IN ACCUM.*/
        IL(CLKO) OH(SYNC,CLK1) OL(CLK2);
        IH(CLKO) IG(DATA=X'AE') OL(CLK1) OH(CLK2);
        IL(CLKO) OG(ADDR=X'0402') OL(SYNC,CLK2) OH(CLK1);
        IH(CLKO) IG(DATA=X'00') OL(CLK1) OH(CLK2);
        IL(CLKO) OG(ADDR=X'0403') OH(CLK1) OL(CLK2);
        IH(CLKO) IG(DATA=X'0A') OL(CLK1) OH(CLK2);
        IL(CLKO) OG(ADDR=X'0A00') OH(CLK1) OL(CLK2);
        IH(CLKO) IG(DATA=X'7A') OL(CLK1) OH(CLK2);
        IL(CLKO) OG(ADDR=X'0404') OH(SYNC,CLK1) OL(CLK2),
        IH(CLKO) IG(DATA=X'8A') OL(CLK1) OH(CLK2);
        IL(CLKO) OG(ADDR=X'0405') OL(SYNC,CLK2) OH(CLK1),
        IH(CLKO) OL(CLK1) OH(CLK2);
```

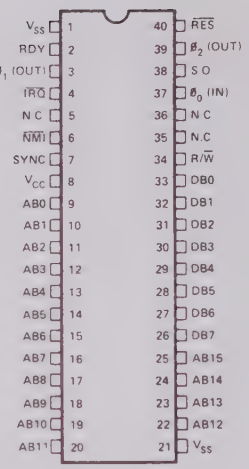


Figure 6-8. Beginning of Test for R6502 Microprocessor.

The HEADer section for this 40-pin device assigns symbolic names to the pins on the processor and shows which are to be treated as INPUTs, OUTPUTs, and as TRIstate. The input lines Ready (RDY), Interrupt Request (IRQ), Non-Maskable Interrupt (NMI) and Set Overflow (SO) may be unconnected in some applications of the processor; if they were declared simply as inputs, DTG would refuse to generate a test if it found any of them disconnected. Instead, they are declared as Default High (.DH) to indicate that they should be treated as high if not connected.

```
/*6502          */

.HEAD;
        .SIZE 40;
        .INPUT(2=RDY,4=IRQ,6=NMI,37=CLK0,38=SO,40=RSET);
        .OUTPUT(3=CLK1,7=SYNC,9,10,11,12,13,14,15,16,17,18,19,20,
                22,23,24,25,34=RW,39=CLK2);
        .TRI(26,27,28,29,30,31,32,33);
        .DH(RDY,IRQ,NMI,SO);
.END HEAD;
```

By combining all 8 data lines as a GROUP, an opcode can be placed on the data bus with the simple DTSL test statement, IG(DATA=X'A9'), which means, apply the bit pattern corresponding to hex A9 to the pins in the GROUP DATA. Both the data and the address buses are declared as GROUPs, by giving the pin numbers on the IC package that connect to them.

```
.DECLARE GROUP ADDR(25,24,23,22,20,19,18,17,16,15,14,13,12,11,10,9);
.DECLARE GROUP DATA(26,27,28,29,30,31,32,33);
```

DTG requires instructions that will let it prevent the 6502 from interfering with tests of other devices. One way to do this is to pull the processor's Ready line low and clock it in, as is done in the 6502 INHibit section. This arrests the processor, so that its outputs will not change state during the test of another device. In this INHibit section the processor is clocked 10 times with Ready low; then DTG is instructed to hold the Ready line low while other devices are tested.

```
.INH (DATA,ADDR,RW,SYNC);
        IL(RDY);
        LOOP=D'10' IL(CLK0);
        IH(CLK0) END LOOP;
        .HOLD(RDY);
.END INH;
```

In using the FORCE, INHibit, and DISABle sections of the device procedure to test other devices on the board, DTG takes care of connecting the required driver/sensors. But in the sections belonging to the test proper (i.e., MAIN and any SPECial sections), driver/sensor connections must be specified, just as in a digital test language test. Thus, the first Input Connect statement (IC) in the 6502 MAIN section connects all of the inputs that are required for the test to begin. (Only those driver/sensors should be connected whose states are known; therefore, the data bus is not yet connected.) If possible, the Ready, NMI, and SO lines are pulled high using PH (Prefer High). The IRQ and Reset lines must be high for the processor to function normally, so these pins states are specified as IH. If DTG finds from the circuit description that it cannot bring these pins high, it will reject the test. The state of the data bus is not known at the beginning of the test, so the No Preference statement, NP, tells DTG to ignore the states of the data lines.

```
.MAIN;

        IC(RDY,IRQ,NMI,CLK0,SO,RSET,DATA) PH(RDY,NMI,SO) IH(IRQ,CLK0)
        IL(RSET) NP(DATA);
```

The first stage of the processor test is the reset sequence that loads an initial address into the program counter and synchronizes it to the tester.

For the 6502 this is easily accomplished by holding the Reset line low (IL(RSET) in the statement above) and clocking it for 10 cycles with the following loop.

```
/*POWER UP CYCLE*/
        LOOP=D'10';
        IL(CLK0);IH(CLK0);
        END LOOP;
```

Once Reset comes high, the processor requires 6 clock cycles before it will fetch the Reset address. During this loop no data is read from memory, so R/W should be found high. Note that even though the statement OH(RW) does not appear inside the loop, the R/W line is in fact tested at each step of the loop because it now belongs to the nail-state context. When in debug, only the last pass in the loop is shown in the debug display. It will continue to be tested until removed from the context by the statement OI(RW).

```
        IL(CLK0);IH(RSET,CLK0) OS(RW) OH(RW);
        LOOP=D'6';
        IL(CLK0);IH(CLK0);
        END LOOP;
```

The 6502 reset address is FFFC, and this address should appear on the address bus on the seventh clock cycle after Reset has gone high. So at this point the address bus is added to the nail context and tested for the presence of hex FFFC. The output clock lines (phase 1 and phase 2) are added to the nail context and are followed through each clock cycle. Before clocking the processor to the fetch state, the test places the low-order byte of the test's starting address, hex 00, on the data bus. It then checks that the address increments to hex FFFD for fetching the high order byte of the reset vector, 04.

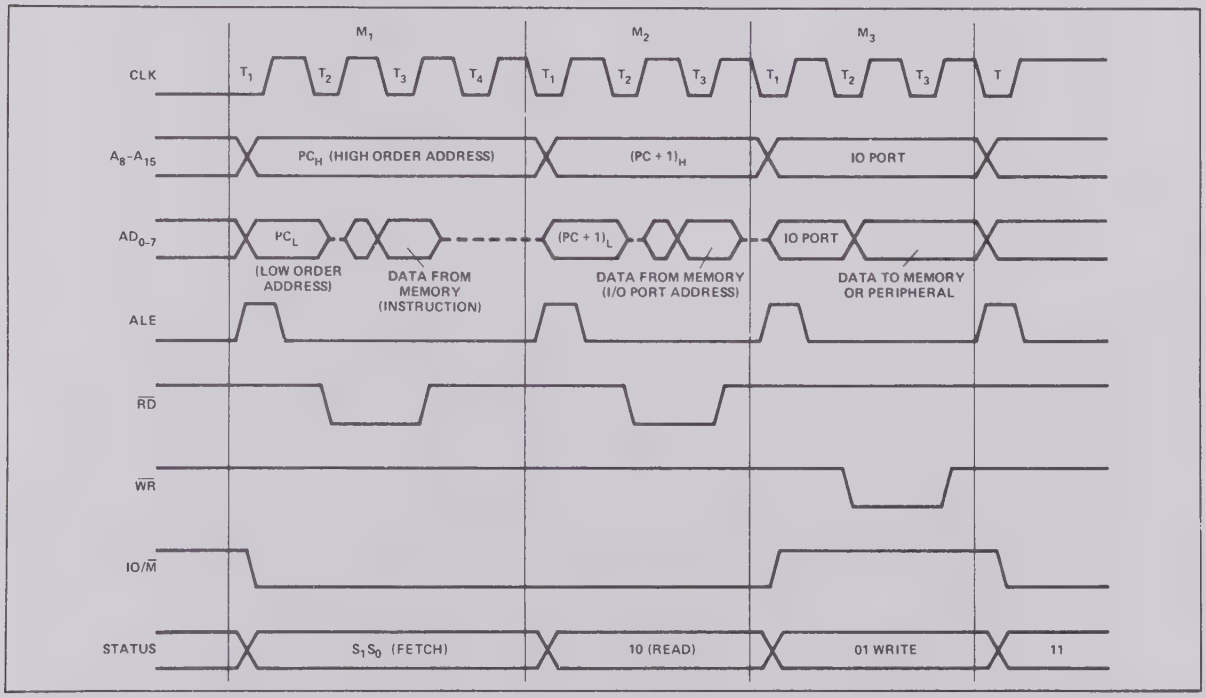
```
    IL(CLK0) OS(ADDR,CLK1,CLK2) OG(ADDR=X'FFFC') OH(CLK1) OL(CLK2);
    IH(CLK0) IG(DATA=X'00') OL(CLK1) OH(CLK2);
    IL(CLK0) OG(ADDR=X'FFFD') OH(CLK1) OL(CLK2);
    IH(CLK0) IG(DATA=X'04') OL(CLK1) OH(CLK2);

/*INSTRUCTION FETCH; CLEAR CARRY FLAG (DONE FOR
  INITIALIZATION PURPOSES)*/

    IL(CLK0) OG(ADDR=X'0400') OS(SYNC) OH(SYNC,CLK1) OL(CLK2);
    IH(CLK0) IG(DATA=X'18') OL(CLK1) OH(CLK2);
    IL(CLK0) OG(ADDR=X'0401') OL(SYNC,CLK2) OH(CLK1);
    IH(CLK0) OL(CLK1) OH(CLK2);
```

Following the reset, the processor ought to place the address 0400 on the address bus for its first instruction fetch. The Sync line should go high just before an opcode is read, so SYNC is added to the nail context and checked for a high at this point. Bringing CLK0 high initiates the instruction fetch, and the instruction CLC (clear carry, hex 18) is placed on the data bus (for the sake of initialization).

The next instruction sequence loads the index register X (opcode AE) from location 0A00 with the value 7A and moves it to the accumulator (opcode 8A). Further instruction sequences (not shown) perform other operations on this value and test both the operations and the data paths traversed by finally writing the resulting value back out to the data bus.

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 0V ±5%, V_{SS} = 0V)

Symbol	Parameter	8085A[2]		8085A-2[2]		Units
		Min.	Max.	Min.	Max.	
t_{CYC}	CLK Cycle Period	320	2000	200	2000	ns
t_1	CLK Low Time (Standard CLK Loading)	80		40		ns
t_2	CLK High Time (Standard CLK Loading)	120		70		ns
t_r, t_f	CLK Rise and Fall Time		30		30	ns
t_{XKR}	X_1 Rising to CLK Rising	30	120	30	100	ns
t_{XKF}	X_1 Rising to CLK Falling	30	150	30	110	ns
t_{AC}	A_{8-15} Valid to Leading Edge of Control[1]	270		115		ns
t_{ACL}	A_{0-7} Valid to Leading Edge of Control	240		115		ns
t_{AD}	A_{0-15} Valid to Valid Data In		575		350	ns
t_{AFR}	Address Float After Leading Edge of $\overline{READ}$ ($\overline{INTA}$)		0		0	ns
t_{AL}	A_{8-15} Valid Before Trailing Edge of ALE[1]	115		50		ns
t_{ALL}	A_{0-7} Valid Before Trailing Edge of ALE	90		50		ns
t_{ARY}	READY Valid from Address Valid		220		100	ns
t_{CA}	Address (A_{8-15}) Valid After Control	120		60		ns

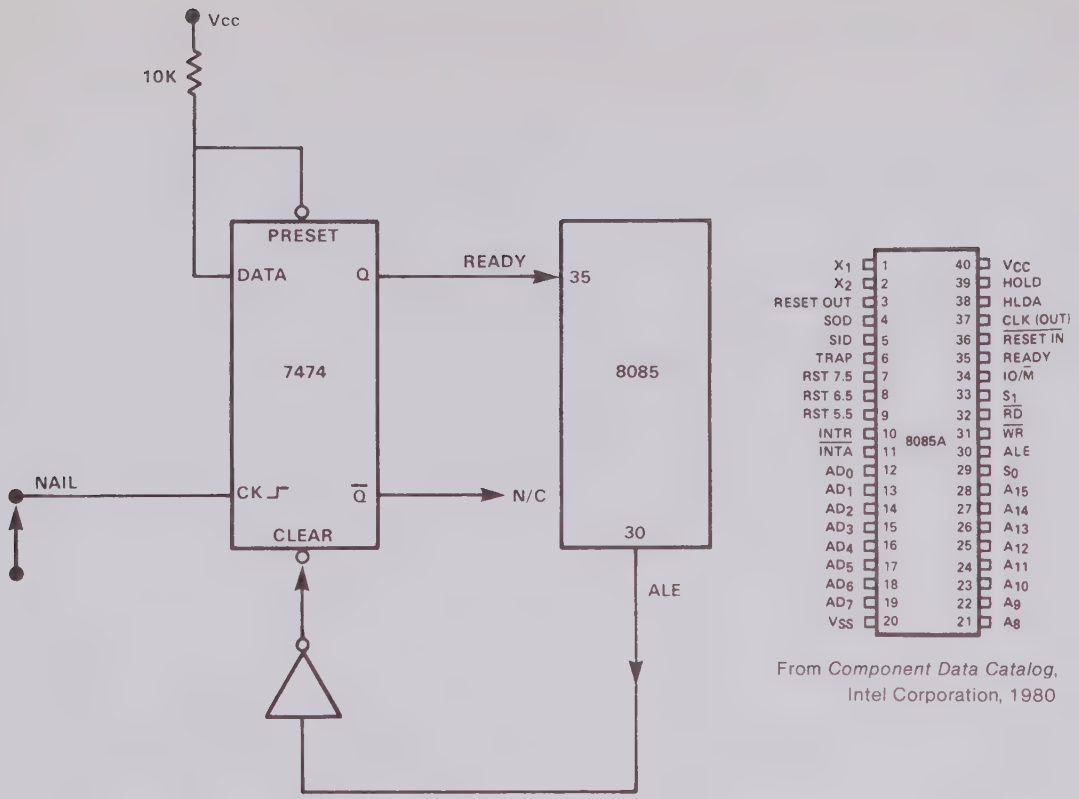


From *Component Data Catalog*, Intel Corporation, 1980

Figure 6-9. Electrical and Timing Characteristics for the I8085.

I8085: Electrical Testability

Because of its multiplexed data/address bus and its timing requirements, the 8085 requires considerably more effort to test than the 6502. Figure 6-9 shows some important timing waveforms and electrical characteristics of the I8085. While it is possible to clock the I8085 externally, the maximum specified clock cycle is 2000 ns, which would require a test rate of 500 kHz (to supply 2 clock edges on each clock cycle). While the processor may function at somewhat lower frequencies than this, the 227X's maximum test rate is 280 kHz, so a synchronous test is not feasible.



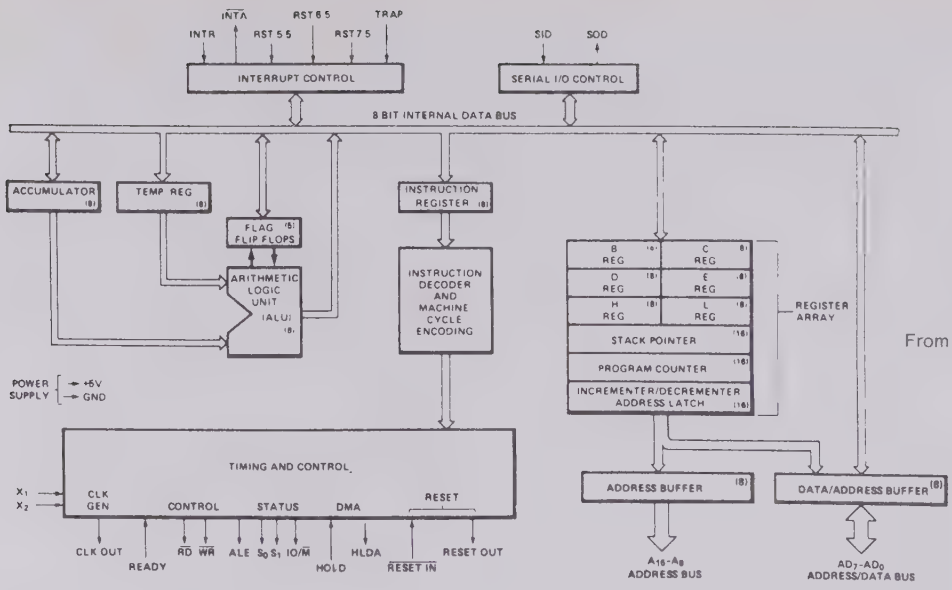
From *Component Data Catalog*, Intel Corporation, 1980

Figure 6-10. Fixture Wiring for the I8085.

Each 8085 processor instruction consists of 1 or more operations (e.g., op code fetch, read, write), called machine cycles. A machine cycle, in turn, consists of 3 or more machine or T states. When the READY line is low, the processor stops after T2 in a state called Twait until the READY line goes high again. Thus, the READY line can be used to arrest a machine cycle between machine states T2 and T3, in Figure 6-9. At this point in the machine cycle the high-order byte of an address is present on the address lines A8-A15 and data have just been latched on to the (multiplexed) data lines or are about to be read from the data lines. Each important operation within an instruction can be individually examined at a speed suitable to the 227X test hardware.

A test nail is assigned to the clock input of the fixture flip-flop, as shown in Figure 6-10. Clocking this nail high and low some number of times has the effect of stepping the 8085 from one Twait state to the next, since it causes READY to go high only as long as it takes the processor to reach the next T state and clear the flip-flop by setting ALE high. While the processor is waiting, the data and address lines can be either driven or sensed. It is assumed that the processor is connected to an on-board crystal or free-running clock generator that drives its X1 and X2 pins; otherwise, an oscillator should be supplied in the test fixture.

The 8085 multiplexes its data bus with the low-order byte of the address bus, so low-order address information is present only during machine state T1. Since we can only halt the processor at the end of state T2, we cannot test the low-order byte of the address bus directly. This is not as serious a difficulty as it might seem at first, however, because the processor hardware that sets up the low-order address can be tested. As is clear from Figure 6-11, testing the data bus tests the same latch that sets up the low-order address. An instruction



From *Component Data Catalog*, Intel Corporation, 1980

Figure 6-11. I8085 Machine Architecture.

such as a CALL that loads an address from the data bus and passes it out to the address bus will verify that both the address buffer and the data/address buffer are functioning properly.

Logical Testability

There is no question of testing the truth table of such a complex device as a microprocessor. However, a test of each of the functional elements in the processor (which would be just as thorough as the truth table test) can be simulated, at least partially, by intelligent choice of the test stimuli. Consider an instruction that, for example, loads the hex value F0 to a register, swaps nibbles, and writes it back to the data bus. This instruction (or sequence of instructions) tests the functionality of all the gates on the path between the data buffer and the ALU, as can be inferred from Figure 6-9. Moving the byte among registers, inverting it, and moving it again would test even more functional elements. The library test shown in Figure 6-12 incorporates a test strategy of this sort, which will be outlined here.

HEADer Section

Here the size and pin-outs of the IC are declared, using the conventional pin mnemonics. Three groups of pins are defined: inputs, outputs, and the bused, tri-stateable pins.

DECLARations

To allow easy manipulation of the data written to and read from the data and high-order address buses, these are declared as the groups DATA and ADDR. In addition, a variable for address calculations, PC, is allocated.

DISABles and INHibits

The processor can be disabled (so that it will not interfere with other ICs while they are being tested) by holding either the Reset pin low, or the

Hold pin high, so DTG is given both options. Likewise, the Hold pin effectively inhibits the processor from changing its outputs while other devices are being tested, so the INHibit section also uses it.

INSERT Macros

Because this library procedure is likely to be used in more than one application and will require long-term maintenance, it is worthwhile to design a set of general INSERTs that can be used flexibly at a number of places in the test procedure. Because the I8085 test is meant for a general-purpose library, it has been written in a highly modular form, rather than being coded 'in-line' in the MAIN section.

The first of these inserts, HCLK, simulates clocking the processor through a machine cycle. As Figure 6-10 shows, pin 35 (=READY) will actually be connected to a flip-flop in the test fixture, rather than to the IC. When READY goes high, the flip-flop raises the processor READY line long enough for it to advance to Twait of the next machine cycle. HCLK does not bring the READY line low again, so it is the responsibility of the calling routines to do this. The insert also instructs the tester to ignore most of the output lines until the processor halts again. The CLUSTer instruction is used to preserve nail context, that is, to restore the pin states and connections that existed before HCLK was executed.

The CODE macro executes a single op code or instruction. However, it also checks results of the previous instruction; hence, every call to CODE actually bridges 2 machine instructions. CODE takes 3 parameters: the op code to be executed, supplied to DATA, and the ADDRess and S0 status left on the bus by the previous instruction. Note that ADDR checks only high-order address bytes; as Figure 6-9 shows, the low-order address byte is lost by the time clock cycle T3 arrives because of multiplexing on the address/data bus. The parameter for S0 is not a pin state but rather a statement, such as OH or OL, that specifies the expected state of S0. (This is more economical than passing a 1-bit GROUP to an OG statement.) CODE first 'cocks the trigger' of the fixture flip-flop by bringing READY low. After placing the op code on the data bus, and checking the address and S0 state left by the previous instruction, CODE calls CLK to execute the op code and advance to the next Wait state.

The WRITE macro is used to examine data that the processor has placed on the data bus (for example, as the result of a STA instruction). The pins in the GROUP DATA are normally treated as inputs and have drivers connected to them. For this insert the drivers are disconnected (via ID), sensors are connected (OS), and the data lines are checked for the bit pattern passed to DATA as the second parameter. The processor's S1, Read and Write pins are also checked for proper operation.

INT1 is used to initiate an interrupt. Where possible the Restart lines RST5, RST6, RST7 are pulled high by giving the processor an EI (enable interrupt) instruction (via CODE). As usual, the high-order program count and the S0 line are checked.

CALL1 performs a restart to hex 37 (op code 'FF').

NMI1 is used to verify the restart instruction.

CALL2 performs a call-on-zero instruction (CALLs if the Z flag is set).

```
/*8085          CPU     */

.HEAD;
        .SIZE 40;
        .INPUT (1=X1,2=X2,5=SID,6=TRAP,7=RST7,8=RST6,9=RST5,10=INT,35=READY,
                36=RESET,39=HOLD);
        .OUTPUT(3=RESOUT,4=SOD,11=INTA,29=S0,30=ALE,33=S1,38=HLDA);
        .TRI    (12=D0,13=D1,14=D2,15=D3,16=D4,17=D5,18=D6,19=D7,21,22,23,24,
                25,26,27,28,31=WR,32=RD,34=IOM);
.END HEAD;

.DECLARE GROUP DATA(D7,D6,D5,D4,D3,D2,D1,D0);
.DECLARE GROUP ADDR(28,27,26,25,24,23,22,21);
.DECLARE BSTRING PC(8);

.DISAB(DATA,ADDR,RD,WR,IOM);
        IL(RESET);
        .HOLD(RESET);
        .END DISAB;
.DISAB(DATA,ADDR,IOM);
        IH(HOLD);
        .HOLD(HOLD);
        .END DISAB;

.INH(RESOUT,SOD,INTA,S0,ALE,S1,HLDA,DATA,ADDR,WR,RD,IOM);
        IH(HOLD);
        .HOLD(HOLD);
        .END INH;

.INSERT HCLK;
        .CLUST;
                IH(READY) OI(#HLDA,INTA,RESOUT,SOD);
                .END CLUST;
        .END INSERT;

.INSERT CODE=3;
        IL(READY) IG(DATA=?1) OG(ADDR=?2) ?3(S0);
        .'HCLK';
        .END INSERT;

.INSERT WRITE=2;
        IL(READY) OG(ADDR=?1) ID(DATA) OS(DATA) OG(DATA=?2)
                OL(S1,WR) OH(RD);
        .'HCLK';
        $OI(DATA) IC(DATA) OL(RD) OH(S1,WR);
        .END INSERT;

.INSERT INT1;
/*EI   ENABLE INTERRUPTS-1 BYTE*/
        $PH(RST5,RST6,RST7) IL(INT) NP(SID);
        .'CODE'(X'FB',PC,OH);

/*NOP   NO OPERATION(3 TIMES)-1 BYTE-VERIFIES 10*/
        .'CODE'(X'00',PC,OH);
        .'CODE'(X'00',PC,OH);
        IL(READY);
        $IH(READY,INT);

/*RST 8   RESTART-3 BYTE-REG INDIRECT-VERIFIES EI*/
        IL(READY) IH(DATA) OH(IOM,RD) OL(INTA) PL(INT);
        $IH(READY);
        IL(READY) OS(DATA) OH(INTA) OL(IOM,WR,S1)
                OG(ADDR=X'55',DATA=X'B0');
        .'HCLK';
        IL(READY) OG(DATA=X'0B');
        .'HCLK';
        .END INSERT;

.INSERT CALL1;
/*RST 8   RESTART-3 BYTE-REG INDIRECT*/
        $OH(S1);
        .'CODE'(X'FF',PC,OH);
        .'WRITE'(X'55',X'B0');
        .'WRITE'(X'55',X'08');
        .END INSERT;
```

Figure 6-12. Library Test Procedure for the I8085. (Page 1 of 4)

```
.INSERT NMI1;
/*NOP   NO OPERATION-1 BYTE*/
        $IC(DATA) IH(TRAP) OL(RD) OH(WR,S1);
        .'CODE'(X'00',X'00',OH);

/*NMI PROCEEDS-VERIFIES RST 8*/
        .'WRITE'(X'55',X'00');
        .'WRITE'(X'55',X'39');
        .END INSERT;

.INSERT CALL2;
/*CZ 24H   CALL IF ZERO(EXECUTES)-5 BYTE-IMMEDIATE*/
        $IC(DATA);
        .'CODE'(X'CC',X'00',OH);
        .'CODE'(X'24',X'00',OL);
        .'CODE'(X'00',X'00',OL);
        $OH(SO);
        .'WRITE'(X'55',X'00');
        .'WRITE'(X'55',X'3B');
        .END INSERT;

.INSERT RESET;  /*THIS INSERT ONLY USED FOR THE FORCES*/
        IL(RESET,READY,TRAP,HOLD);
        IH(READY);
        IH(RESET);
        .END INSERT;


.INSERT FORCE=1;        /*THIS INSERT ONLY USED FOR THE FORCES*/
        IL(READY) IG(DATA=?1);
        IH(READY);
        .END INSERT;

.LFORCE(ADDR,RD,IOM);
        .'RESET';
.END LFORCE;

.HFORCE(WR);
        .'RESET';
.END HFORCE;

.LFORCE(DATA,WR);
        .'RESET';
        .'FORCE'(X'9F');
        .'FORCE'(X'77');
        IL(READY);
.END LFORCE;

.HFORCE(ADDR,DATA,RD,IOM);
        .'RESET';
        .'FORCE'(X'9F');
        .'FORCE'(X'2F');
        .'FORCE'(X'D3');
        .'FORCE'(X'FF');
        IL(READY);
.END HFORCE;

.MAIN;
.SAY"8085 REQUIRES DEVICE ADAPTER HARDWARE TO BE TESTED.  PLEASE
     CONSULT APP NOTE FOR THIS INFORMATION.";

/****RESET PROCESSOR****/

        IC(READY,RESET,RST5,RST6,RST7,SID,INT,TRAP,HOLD)
           IL(RESET,READY,TRAP,HOLD) PL(RST5,RST6,RST7,SID) NP(INT);
        IH(READY) OS(RESOUT) OH(RESOUT);
        IH(RESET) OI(RESOUT);
        LET PC=D'0';
        OS(INTA,SOD,ADDR,RESOUT,RD,WR,IOM,SO,S1,HLDA,ALE)
            OH(WR,SO,S1,INTA) IC(DATA) OL(RESOUT,HLDA,ALE,RD,IOM,SOD);
```

Figure 6-12. Library Test Procedure for the I8085. (Page 2 of 4)

```
/****START INSTRUCTIONS****/

/*(1)RIM   READ INTERRUPTS MASK-1 BYTE*/
       FIRST:  .'CODE'(X'20',PC,OH);

/*(2)STA A500H   STORE ACC DIRECT-4 BYTE-DIRECT-VERIFIES 1,2*/
        $PH(RST5,RST6,RST7,SID);
        .'CODE'(X'32',PC,OH);
        .'CODE'(X'00',PC,OL);
        .'CODE'(X'A5',PC,OL);
        IL(READY) OS(DATA) OL(WR,S1,D3) OH(RD,SO,DO,D1,D2) OG(ADDR=X'A5')
          OV(D4=FIRST:RST5,D5=FIRST:RST6,D6=FIRST:RST7,D7=FIRST:SID);
        .'HCLK';

/*(3)RIM   READ INTERRUPT MASK-1 BYTE*/
        $OH(WR,S1) OL(RD) IC(DATA);
        SECOND: .'CODE'(X'20',PC,OH);


/*(4)LXI H,5601H   LOAD REG PAIR IMMEDIATE-3 BYTE-IMMEDIATE*/
        $PL(RST5,RST6,RST7,SID);
        .'CODE'(X'21',PC,OH);
        .'CODE'(X'01',PC,OL);
        .'CODE'(X'56',PC,OL);

/*(5)MOV M,A   MOVE ACC TO MEMORY-2 BYTE-REG INDIRECT-VERIFIES 3,4,5*/
        .'CODE'(X'77',PC,OH);
        IL(READY) OS(DATA) OL(WR,S1,D3) OH(RD,SO,DO,D1,D2) OG(ADDR=X'56')
          OV(D4=SECOND:RST5,D5=SECOND:RST6,D6=SECOND:RST7,D7=SECOND:SID);
        .'HCLK';

/*(6)SPHL   MOVE HL TO STACK POINTER-1 BYTE-REGISTER*/
        $OH(S1,WR) OL(RD) IC(DATA);
        .'CODE'(X'F9',PC,OH);

/*(7)XCHG   EXCHANGE HL WITH DE-1 BYTE-REGISTER*/
        $PH(INT);
        .'CODE'(X'EB',PC,OH);

/*(8)IN FOH   INPUT TO ACC FROM PORT(D5H)-3 BYTE-DIRECT*/
        .'CODE'(X'DB',PC,OH);
        .'CODE'(X'F0',PC,OL);
        $OH(IOM,S1);
        .'CODE'(X'D5',X'F0',OL);

/*(9)MOV B,A   MOVE ACC TO REG B-1 BYTE-REGISTER*/
        $OL(IOM);
        .'CODE'(X'47',PC,OH);

/*(10)SIM   SET INTERRUPT MASK-1 BYTE*/
        IL(READY) IG(DATA=X'30');
        $IH(READY);

/*(11)LDAX D   LOAD ACC INDIRECT(55H)-2 BYTE-REG INDIRECT-VERIFIES 7*/
        $OH(SOD);
        .'CODE'(X'1A',PC,OH);
        .'CODE'(X'55',X'56',OL);

/*(12)MOV C,A   MOVE ACC TO REG C-1 BYTE-REGISTER*/
        .'CODE'(X'4F',PC,OH);

/*(13)PUSH B   PUSH BC REG PAIR-3 BYTE-REG INDIRECT-VERIFIES 6,8,9,11,12,13*/
        .'CODE'(X'C5',PC,OH);
        .'WRITE'(X'56',X'D5');
        .'WRITE'(X'55',X'55');

/*(14)RRC   ROTATE ACC RIGHT THROUGH CARRY-1 BYTE*/
        .'CODE'(X'0F',PC,OH);

/*(15)CC AFFCH   CALL IF CARRY FLAG IS SET(EXECUTES)-5 BYTE-IMMEDIATE
             VERIFIES 14*/
        .'CODE'(X'DC',PC,OH);
        .'CODE'(X'FC',PC,OL);
        .'CODE'(X'AF',PC,OL);
        $OH(SO);
        .'WRITE'(X'55',X'00');
        .'WRITE'(X'55',X'16');
```

Figure 6-12. Library Test Procedure for the I8085. (Page 3 of 4)

```
/*(16)OUT OFH   OUTPUT ACC TO PORT OFH-3 BYTE-DIRECT-VERIFIES 15,16*/
        .'CODE'(X'D3',X'AF',OH);
        .'CODE'(X'OF',X'AF',OL);
        $OH(SO,IOM);
        .'WRITE'(X'OF',X'AA');

/*(17)POP H   POP HL REG PAIR-3 BYTE-REG INDIRECT*/
        $IL(INT) OL(IOM);
        .'CODE'(X'E1',X'AF',OH);
        .'CODE'(X'FF',X'55',OL);
        .'CODE'(X'FE',X'55',OL);

/*(18)ADD M   ADD ACC TO MEMORY-2 BYTE-REG INDIRECT-VERIFIES 17*/
        .'CODE'(X'86',X'AF',OH);
        .'CODE'(X'57',X'FE',OL);
        LET PC=X'B0';

/*(19)PUSH PSW   PUSH PROCESSOR STATUS WORD-3 BYTE-REG INDIRECT-
                 VERIFIES 18,19*/
        .'CODE'(X'F5',PC,OH);
        .'WRITE'(X'55',X'01');
        IL(READY) OS(DATA) OG(DATA=X'11') OI(D1,D3,D5) OG(ADDR=X'55')
          OL(WR,S1) OH(RD);
        .'HCLK';
        $OI(DATA) IC(DATA) OH(WR,S1) OL(RD);

/*(20)XRA C   EXCLUSIVE OR ACC WITH REG C-1 BYTE-REGISTER*/
        .'CODE'(X'A9',PC,OH);

/*(21)SBI 54H   SUBTRACT IMMEDIATE WITH BORROW-2 BYTE-IMMEDIATE*/
        .'CODE'(X'DE',PC,OH);
        .'CODE'(X'54',PC,OL);

/*(22)JNZ 00H   JUMP IF ACC IS NOT ZERO-2 BYTE-IMMEDIATE-VERIFIES 20,21,22*/
        .'CODE'(X'C2',PC,OH);
        .'CODE'(X'00',PC,OL);

/*(23)INTERRUPT OR CALL*/
        .CHOOSE I(INT1,CALL1);

/*(24)NMI OR CALL*/
        .CHOOSE I(NMI1,CALL2);

/*(25)HLT HALT THE PROCESSOR-1 BYTE-VERIFIES 25*/
        .'CODE'(X'76',X'00',OH);
        IL(READY) OI(#SO,S1,SOD,RESOUT) OL(SO,S1);
        IH(HOLD) OI(#SOD,RESOUT);
        OS(HLDA) OH(HLDA);
        ID(#) OI(#);
        .END MAIN;
```

Figure 6-12. Library Test Procedure for the I8085. (Page 4 of 4)

HFORCE and LFORCE Macros

The first LFORCE section allows ATG to cause the processor to force the address bus, the read line, and the I/O line low during bus test. It calls the Reset insert to do this. The other LFORCEs use the same insert, as well as the Force insert.

The HFORCEs function similarly.

MAIN Section

It is from the .MAIN (or .SPECial) section that DTG produces the body of a component test, after drawing the necessary disable and inhibit procedures for surrounding components from the library entries for those components. The test first resets the processor and then executes a number of instructions, described in the comment preceding each subsection, using the 'CODE' insert. The test's strategy is to move a byte or word of data among registers and to and from memory, using various addressing modes and verifying that both byte and word operations execute properly. In addition, instructions of varying length -- 1 to 5 bytes -- are tested. In this way, a good sampling of both the processor's internal data paths and its control logic is tested in a reasonable amount of time.

6.2.8 Developing a Test Procedure for the SpeedPlus Subsystem

As mentioned in para 6.2.6, procedures using the SpeedPlus subsystem should be provided only for components that can be tested considerably more effectively on the SpeedPlus hardware. Whenever possible, a new library procedure should be based on an existing library procedure for the same or a similar component to save development time.

Some library test procedures supplied by GenRad for VLSI devices contain sections that can make use of the SpeedPlus capabilities; they usually also contain tests designed to run on the standard digital test hardware, to retain compatibility with systems not having the SpeedPlus option, and to allow testing when other components on the board have preempted the SpeedPlus resources. Generally, the auxiliary sections of a library — .DISable, .HFORCE, etc. — are written for standard driver/sensor nails; where SpeedPLus auxiliary sections are included (e.g., .DISable USING), they follow the standard section, so that DTG will usually write the standard procedure into the test program, regardless of whether the component is assigned to standard or to SpeedPlus test nails (in the circuit description, or .CKT, file). By contrast, a SpeedPlus .MAIN section must always be placed ahead of the standard section in the library, and consequently DTG will always write a SpeedPlus test for a component that is assigned to the required SpeedPlus nails. It will only select the standard test if the requirements for the SpeedPlus test are not met.

The main difference between a SpeedPlus test and a standard component test lies in the much greater control over test timing parameters that the SpeedPlus subsystem allows. While test steps are programmed in the same way, the steps used to toggle clock pins and maintain tester synchronization can usually be eliminated from the SpeedPlus test. In their place, the SpeedPlus clock module will be used to establish steady-state clock signals, and DTG writes an appropriate SET CLOCK statement into the test program preceding the burst in which the component is tested. Like the BURST parameters IST and ITT, the SET CLOCK parameters define drive and sense timing. Additional parameters allow precise control of a number of other test events, as well. A SpeedPlus trigger mechanism can be activated within any test step by describing trigger nail states that must occur before the rest of the test statement can execute. The test statements used for programming the SpeedPlus facilities are described in the Test Programming manual, para 4.4 and 6.2. The procedures for debugging and verifying operation of a SpeedPlus test are described in the Test Set Preparation manual, para 9.3.

To use these special hardware facilities effectively, the designer of a SpeedPlus component test must first study the component's timing specifications carefully to establish a range of acceptable clock frequencies. (As noted in para 6.2.6, the lowest acceptable frequency for a clocked component at room temperature is often considerably lower than the lowest rated frequency.) This range should be made as wide as possible, so that DTG will have as much freedom as possible to activate several devices at once using a common set of clock parameters, as, for example, during bus tests. The range of permitted clock frequencies is then specified in the .FREQ statement in the procedure header.

In many cases, however, a device requires a special waveform or more than one clock phase, and the programmer will need to write one or more explicit clock definitions, using the DTSL construct .CLOCK. Bear in mind that any disable, inhibit or force sections requiring the SpeedPlus clock may have to share the clock with one or more other components. For example, a peripheral device such as a CRT controller may need to be clocked to disable it during

testing of a microprocessor. Both the CRT controller and the microprocessor may require clock signals, and the library descriptions of the 2 components may specify different clock parameters. DTG is generally unable to resolve such conflicts, and it will in fact reject auxiliary (.DISABle or .INHibit) sections whose clock requirements are different from those of the component being tested. Such conflicts should be resolved by the test programmer by writing a SET CLOCK statement into the .ATX file associated with the circuit description.

Once basic clock requirements have been established, a strategy must be developed for drive, sense, and trigger timing relative to the clock signal(s). Careful examination of the component's timing diagrams will usually show that all pins change state or latch in applied data at certain well-defined times in relation to the clock signal -- for example, at a certain interval after every rising clock edge. Using the timing diagrams, it will be possible to define times at which data from the component can be latched in by the test system's sensors, as well as times where the test drivers should change state. The component's timing requirements should then be compared to the DTSL default timing parameters, described in para 6.3.3 under CLOCK.

If it is found that all parts of the procedure can operate with all the default clock and test timing parameters, a .FREQ statement should be entered in the .HEADer section, to allow DTG maximum freedom in writing a SET CLOCK statement for the test program. Briefly, the default clock parameters specify a symmetrical, square-wave clock signal in the allowed frequency range, with drive strobes applied on every clock edge (rising and falling), sense strobes applied 50 to 200 ns ahead of each drive strobe, and trigger nail states latched just ahead of every -- or just the first -- drive strobe . If any timing parameters different from the defaults are required to test the component, one or more .CLOCK statements must be entered in the procedure. Each section of the procedure will then reference a .CLOCK statement, and DTG will only write the section into a test when it is able to use the corresponding SET CLOCK statement.

In addition to taking over the clocking function from the test program, the SpeedPlus subsystem is also designed to simplify the testing of microprocessors' irregular machine operations. A microprocessor operation sometimes extends over many clock cycles in which nothing of interest to the tester occurs and it can start or finish at an imprecisely known time (for example, in an asynchronous bus transaction). Consequently, the writer of a test procedure may find there are machine states that need to be tested only after some number of intermediate states have elapsed, during which the test program can suspend itself. A test step can be suspended by including in it a trigger condition, consisting of the terms TH(<pin-list>) and TL(<pin-list>) or TG(<group-name>). The step will not execute until the pins named in the trigger pin lists are in the required states, so there is no need to write 'filler' steps or wait loops to keep the program in step with the microprocessor operation. Note that a given trigger pattern, such as TH(1,2) TL(3,4), can be used any number of times; however, only 15 distinct trigger patterns may appear in a burst. (Figure 6-15 shows some examples of how trigger patterns can be used to condense portions of a microprocessor test (MC68000). Refer to the Test Programming manual, para 4.3, for a detailed discussion of the SpeedPlus trigger mechanism.)

6.2.9 Structure of a SpeedPlus Model

A library procedure meant to use the SpeedPlus subsystem should normally provide procedures for standard digital testing as well, so that standard tests can be written when SpeedPlus nails to test the component are not available. A

library procedure that uses the SpeedPlus subsystem in any way is identified by the statement .FTEST USING in its header. Each section of the procedure meant to execute on the SpeedPlus facilities is then identified by its own USING term, which lists the component pins requiring SpeedPlus nails in that section. If DTG finds that the component pins requiring fast driver/sensor and timing nails are not assigned to SpeedPlus nails, it will reject the section; it will also reject an auxiliary section whose clock requirements conflict with the clock required by the UUT.

HEADer and CLOCKs

A library procedure that requires resources from the SpeedPlus subsystem must declare its requirements in the .HEAD statement. Statements used in addition to the .HEAD parameters required for standard testing inform DTG about the SpeedPlus nails and timing parameters that the procedure may need. These statements include

```
.FTEST USING ( F= <'high-speed' pins>, C=<clock pins>, T=<trigger pins>);
.FREQ <lowest permitted speed> TO <highest permitted speed>;
```

In addition, if timing parameters (drive-sense timing, triggering, or clock waveforms) different from the default values are required, these must be defined in one or more clock specifications of the form

```
.CLOCK <name>; <clock parameters>; .END CLOCK;
```

.FTEST USING lists all the component pins that, at any point in the procedure, may require SpeedPlus test or timing nails. The terms F, C, and T give the kinds of SpeedPlus nails they will need; "F" stands for "fast," or SpeedPlus driver/sensors; "C" stands for "clock nails"; and "T" stands for "trigger nails." Before it writes a burst, DTG must, in addition, compute a set of timing parameters to be used by all FAST blocks in the burst, and it must write them into a SET CLOCK statement ahead of the burst. The component's timing requirements can be specified in the library procedure in 1 of 2 ways. If the procedure contains 1 or more .CLOCK statements, DTG will use the data in 1 of these statements to write out the component's timing requirements. However, if the HEADer contains the statement .FREQ, DTG will create an implicit, unnamed .CLOCK statement having a frequency in the allowed range, with all other parameters defined by the DTSL default clock values. DTG will then use this implicit clock to write a SET CLOCK statement into the test program (.DTP) file.

Each section of the library procedure that requires SpeedPlus resources is introduced by the name of the section and the term USING. (Note that the syntax for this USING term is slightly different from that in the .HEADER in that it specifies only the nails used in the sections, and it can reference a .CLOCK statement by its name). If all sections of the procedure can operate using the DTSL implicit clock (defined only by the .FREQ statement in the .HEADer section), the USING clause for each section takes the form,

```
<section-name> USING (C=<pin-list>, T=<pin-list>, F=<pin-list>);
```

However, the clocking and test timing required for various sections may differ (e.g., it may be possible to carry out an LFORCE at a lower clock frequency than is required for the main test). In this case, it is not necessary for the .HEADer section to contain the .FREQ statement; instead, .CLOCK statements will be placed after the .HEADer section to define the required timing parameters. The USING clause for each section then references a CLOCK statement that gives

the required parameters, and every USING clause takes the form

<section-name> USING (<clock-name>, T=<pin-list>, F=<pin-list>);

In this second form, the clock pin connections are drawn from the AT portion of the .CLOCK statement referred to by <clock-name>.

The .CLOCK section has essentially the same form and default parameters as the SET CLOCK statement in the test language, except that it is named and lists component pin connections rather than nail numbers. A library procedure can contain any number of .CLOCK statements, but each section (e.g., .DISABle, .LFORCE, .SPECial, .MAIN) can reference only one clock. Morever, when DTG writes a burst containing FAST blocks, it takes the clock parameters specified by the .MAIN USING (or .SPECial USING) section -- via either the .FREQ specification or a named .CLOCK -- and tries to accommodate them to the timing requirements of any other components that must be disabled or forced during the burst. DTG can write only one SET CLOCK statement for a burst; hence, the timing requirements of all sections DTG choses must be compatible, and any incompatible auxiliary sections will be rejected. An auxiliary section is much more likely to be found incompatible when it names a clock specification than when it references DTG's implicit (default) clock specification.

A .CLOCK statement specifying that 2 clock phases must be supplied to a component's clock pins, named PHASE1 and PHASE2, at 1 Mhz (with other parameters defaulting), would appear as

```
.CLOCK CLK1;
 AT (CDA=PHASE1 : CDB=PHASE2)
 F=1MEG
 NINC=4
 SIGNAL(CDA=H0,L1 : CDB=H2,L3);
.END CLOCK;
```

DISABles, INHibits, FORCEs

A library procedure for the SpeedPlus subsystem can contain both standard and SpeedPlus versions of both the UUT test (.MAIN or .SPECial section) and the auxiliary test information (.DISABle, .INHibit, .LFORCE, etc.). Where a component's timing requirements allow, all of the required auxiliary test information should first be entered as sections written for the standard test hardware. Following each standard INHibit, DISABle, and force section, corresponding sections for execution on the SpeedPlus hardware can be supplied, as described in para 6.2.6. These SpeedPlus sections are distinguished from the standard sections by the USING clause, giving clock data and listing the component pins that require SpeedPlus resources. For example,

```
.INH (DTACK, WRITE) USING (CLK1, F=<pins>);
.END INH;
```

or

```
.LFORCE (DATA) USING (C=PHASE1, F=<pins>);
.END LFORCE;
```

By placing the SpeedPlus sections after the corresponding standard sections, the programmer ensures that DTG will always evaluate the standard auxiliary sections first and use them whenever possible. Typically, a component can be disabled by forcing just a few pins to a state that simply needs to be held during the testing of another component, and there is no need to assign

SpeedPlus driver/sensors for this purpose. If a component needs to be clocked while the disabling pins are held, the clock pin connection will be specified in a USING clause, but no FAST block will be included. The USING term will cause DTG to add the required clock statement ahead of the burst and to connect the component's clock pin to a clock driver; the pins that control the disable or force will be connected to standard driver/sensors.

SPECial and MAIN Sections

Like the auxiliary sections, .SPECial and .MAIN sections can be written with a USING clause, which notifies DTG that a SET CLOCK statement must precede the burst. If the SpeedPlus nails required by the section are not assigned to the component in the circuit description, the section will be not be used.

As with the auxiliary sections, SpeedPlus .SPECial and .MAIN sections should be accompanied by equivalent sections written for the standard hardware whenever possible. However, the .MAIN USING section should precede the standard .MAIN section. DTG will select the .MAIN USING section if the SpeedPlus resources it requires are assigned to the component, but it will always use a standard .MAIN section as soon as encountered. A .MAIN USING section included after a standard .MAIN section will never be used.

6.2.10 Some SpeedPlus Testing Strategies

The SpeedPLus subsystem provides a number of highly specialized capabilities for microprocessor testing, yet it is also compatible with the features of the standard digital test hardware. Because test systems with the SpeedPlus subsystem have only a limited number of SpeedPlus test nails, SpeedPlus library procedures should be designed to use these nails only where they are genuinely needed, Effective library tests can be written that use both the standard test nails and a limited number of SpeedPlus nails. Wherever possible, both standard and SpeedPlus versions of a test should be included in a general-purpose library procedure, so that test programmers can make use of the available system resources in the most effective way.

Using the SpeedPlus Trigger Mechanism

A simple example, adapted from the 227X library test procedures for the MC68000 microprocessor, will clarify the use of trigger statements in FAST sections. The MC68000 typically requires a number of clock cycles to perform a bus operation, such as a read or write. Without the SpeedPlus hardware, the test program must provide 2 test vectors for each clock cycle, whether or not any other inputs or outputs need to be tested during the cycle. Using the SpeedPlus clock and trigger facilities, however, a test program can be written that drives and senses only at the relevant points in the bus cycle.

Consider first a procedure for testing the processor's "read" operation with the standard hardware. When a test program causes the processor to initiate a read bus cycle (of an instruction or data word), it must first check that the processor places the correct address on the address bus. At the proper time the tester must then supply the expected data word, by driving the data pins to the desired states. As the timing diagram for the read cycle in Figure 6-13 shows, a read bus cycle requires 4 clock cycles (machine states S0 to S7). During the first clock cycle, the processor places an address on the address bus. The processor expects data to become valid by the third clock cycle, and latches it in the fourth clock cycle. The tester should remove the data from the data bus at the end of the fourth clock cycle.

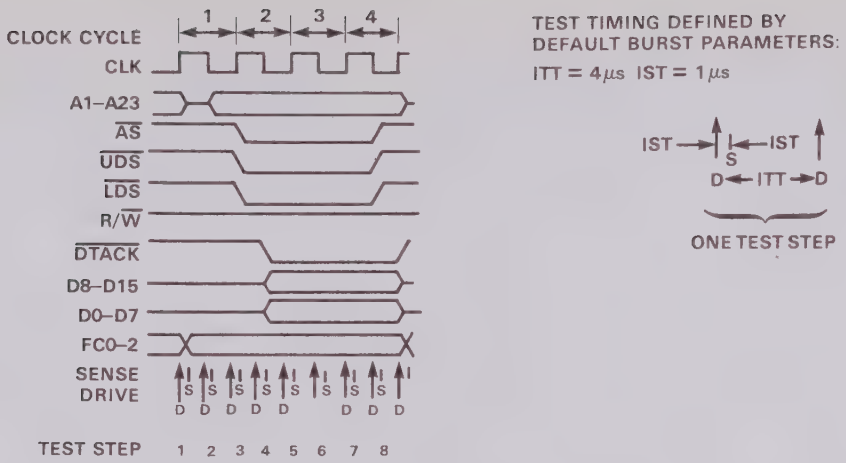


Figure 6-13. Read Cycle Timing for Read, MC68000 Microprocessor. Standard Test Events.

To save space in this example, test nails attached to the address bus are shown simply as "<addr>"; data nails are identified as a group as "<data>" or as "<1-data>" for the high bits and "<0-data>" for the low bits. "IH(<1-data>) IL(<0-data>)" means that some of the nails on the processor's data pins are driven high and that the rest are driven low. The address and data strobes are lumped as "<ds>". CLK stands for the clock input, RW is Read/Write and DTACK is the asynchronous bus handshaking line (Data Transfer Acknowledge). Notice that altogether 3 test vectors are used here for nothing but clocking the microprocessor.

```
/* Clock cycle 1 */
/* Processor applies an address to the address bus */
        IH(CLK)  IH(DTACK);
        IL(CLK)  OS(<addr>,RW) OH(RW) OH(<1-addr>) OL(<0-addr>);

/* Clock cycle 2 */
/* Tester applies data input to processor's data inputs */
        IH(CLK)  OL(<ds>);
        IL(CLK)  IL(DTACK) IC(<data>) IH(<1-data>) IL(<0-data>);

/* Clock cycle 3 */
        IH(CLK);
        IL(CLK);

/* Clock cycle 4 */
/* Tester checks end of bus cycle */
        IH(CLK);
        IL(CLK)  OH(<ds>);
```

A test written for the SpeedPlus hardware, by contrast, relies on the clock generator for the alternating highs and lows applied to the clock pin, and it skips the third and part of the fourth clock cycles. This economy is achieved by using trigger commands to wait for the processor's address strobe, AS, to assume the states corresponding to the beginning and end of the data transfer. The tester interacts with the microprocessor in much the same way as an asynchronous peripheral, and, in fact, microprocessor tests on the SpeedPlus

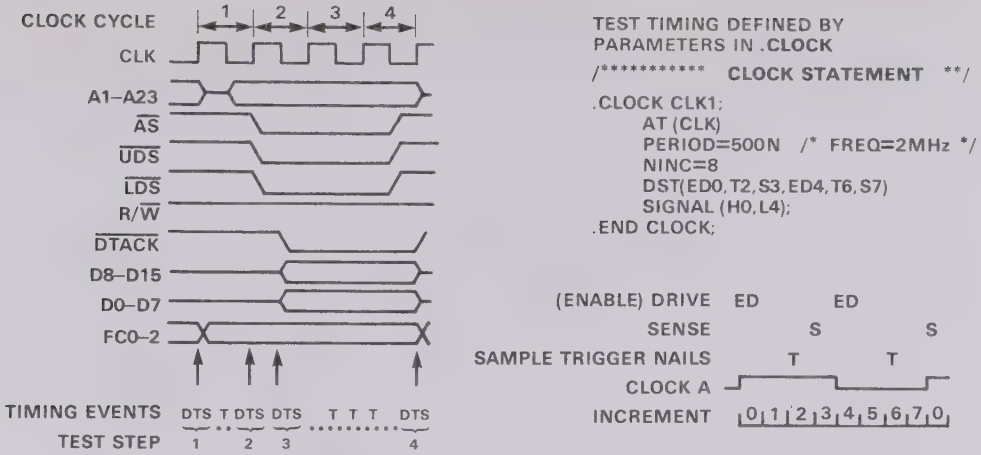


Figure 6-14. Bus Cycle Timing for Read, MC68000 Microprocessor. SpeedPlus Test Timing.

system can often be conceived as sets of operations that might be performed by the processor's intended peripherals.

To see how the driver, sensor, and trigger strobes are coordinated with the clock, examine the clock statement shown in Figure 6-14. (Refer to the Test Programming manual, para 6.3, for a description of the SET CLOCK statement.)

Using this clock, the test of the read bus cycle executed on the SpeedPlus hardware becomes much shorter, and the test program interacts with the processor in nearly the same way as an addressed peripheral. The clocking sequence IH(CLK); IL(CLK) is, of course, no longer required, and synchronization to the bus cycle is obtained by waiting on the low and high transitions of the address and data strobes, much as any asynchronous peripheral would do. Note that there are fewer test steps, and that no test steps at all have been written for the third clock cycle. The test sequence looks like this (address and data nails are again simply indicated as "<addr>" and "<data>"):

```
/* Clock cycle 1 */
/* Ignore all outputs except RW on first half-cycle */
            IH(DTACK) ID(<data>) OI(<addr>) OH(RW);

/* Clock cycle 2 */
/* Wait for address strobe to be asserted */
            TL(<ds>)
/* then check value of address and RW */
                  OS(<addr>,RW) OH(RW) OH(<1-addr>) OL(<0-addr>);

/* Then send bus acknowledgement; supply data */
            IL(DTACK) IC(<data>) IH(<1-data>) IL(<0-data>);

/* Clock cycle 3 */
            /* nothing to do; processor latches in data */

/* Clock cycle 4 */
/* Wait for end of address-data strobe */
            TH(<ds>)
/* and finish up by disconnecting sensors */
                  OI(<data>, RW, <addr>);
```

Figure 6-14 shows in detail the timing of test events relative to the clock signal. The most important difference from the sequence in Figure 6-13 is that the test timing is irregular -- 2 test events occur in the second clock cycle, and only 1 each in the first and fourth cycles.

Figure 6-15 is condensed from the digital test library procedure for the 68000; it illustrates how test sequences written for the standard hardware can be simplified and shortened when they are written for the SpeedPlus hardware. Notice, for example, that the standard test requires 2 separate read inserts to allow for the microprocessor's variable timing. The second insert, READSYNC, must be used when the processor requires additional time to place a valid address on the bus. In the SpeedPlus version, the tester simply monitors the address and data strobes, grouped together as DS, via the term TL(DS) in the second step. The step waits until they are asserted low, then checks for the expected address. To acknowledge that the address has been received, the tester lowers DTACK and asserts the data the processor is to read. It then waits until the DS pins go high -- via TH(DS) -- to complete the operation.

The SpeedPlus write operation proceeds in a similar way, taking advantage of the handshaking signals provided by the processor. However, the first trigger waits only on the address strobe line, AS, since the data strobes are not asserted until later, when the data are valid.

Normally, loops are not required for timing purposes in a SpeedPlus test. The reset inserts in Figure 6-15, however, show how LOOP and FLOOP constructs are used in both the standard and the SpeedPlus versions of the 68000 test to produce the delay required by the processor during the start of a reset operation. While the standard test needs to apply clock pulses during the loop -- via $IH(CLK); $IL(CLK); -- the SpeedPlus test does not need to perform any test operations at all.

More Complex Timing Patterns

With some components it may be desirable to test for a trigger condition at one point in the timing cycle and store the result of the trigger comparison for one or more clock cycles before executing a test step. In this situation, a test program must wait on some event that occurs early in a bus cycle; no test vectors, however, need to be applied until some number of drive-sense times later. One way to provide the required interval is to write a 'dummy' test step at the point where the trigger will occur and to follow it with as many further dummy steps as are needed to advance to the point where testing should resume. However, if many such 'dummy' steps must be inserted in the program, it may conceivably become too long to execute in a single burst.

A more efficient test would make use of a timing cycle that samples the trigger nail states 1 or more drive-sense pairs ahead of the enable drive time (ED), for example,

```
DST(T1,D2,S3, D4,S5, ED6,S7)
```

Test steps containing no trigger terms will execute on all the D,S drive-sense pairs. However, a test step that contains trigger terms will be suspended until ED6 occurs. No driver strobes are sent, and no test vectors are executed, until the driver enable strobe.

```
/*   68000      CPU,  16-BIT, 64 PINS      VERSION - 1F   */
/*   MOTOROLA Mask - CC1 and GN7, SIGNETICS Mask - C4164  */
/*   SPEEDPLUS   and       STANDARD      TEST             */
/*   ALIASES: MC68000, MK68000                            */
/*   THIS TEST REQUIRES ALL PINS TO BE DRIVEABLE          */
/*   SPEEDPLUS TEST REQUIRES:   55 FAST Nails
                                 1 CLOCK Nail
                                 4 TRIGGER Nails          */

.HEAD;
        .SIZE 64;
        .INPUT (10=DTACK,12=BGACK,13=BR,15=CLK,17=HALT,18=RESET,21=VPA,
                22=BERR,23=IPL2,24=IPL1,25=IPL0);
        .OUTPUT (11=BG,20=EN);
        .TRI (1=D4,2=D3,3=D2,4=D1,5=D0,6=AS,7=UDS,8=LDS,9=RW,19=VMA,26=FC2,
                27=FC1,28=FC0,29=A1,30=A2,31=A3,32=A4,33=A5,34=A6,35=A7,
                36=A8,37=A9,38=A10,39=A11,40=A12,41=A13,42=A14,43=A15,
                44=A16,45=A17,46=A18,47=A19,48=A20,50=A21,51=A22,52=A23,
                54=D15,55=D14,56=D13,57=D12,58=D11,59=D10,60=D9,61=D8,
                62=D7,63=D6,64=D5);
        .FTEST USING (F=1-5 9-13 17-19 21-48 50-52 54-64,C=15,T=6-8 20);
        /*  VCC= 14,49     GND= 16,53  */
 END HEAD;

/* PIN GROUP DECLARATIONS */

    DECLARE GROUP ADDR(A23,A22,A21,A20,A19,A18,A17,A16,A15,A14,A13,A12,A11,
                       A10,A9,A8,A7,A6,A5,A4,A3,A2,A1);
    DECLARE GROUP LADDR(A15,A14,A13,A12,A11,A10,A9,A8,A7,A6,A5,A4,A3,A2,A1);
    DECLARE GROUP HADDR(A13,A14,A15,A16,A17,A18,A19,A20,A21,A22,A23);
    DECLARE GROUP DATA(D15,D14,D13,D12,D11,D10,D9,D8,D7,D6,D5,D4,D3,D2,D1,D0);
    DECLARE GROUP FC(FC2,FC1,FC0);
    DECLARE GROUP INTC(IPL2,IPL1,IPL0);
    DECLARE GROUP DS(AS,UDS,LDS);

/* ADDR,CONTROL AND DATA */

   DECLARE BSTRING PC(24),
   DECLARE BSTRING WORD(4);
   DECLARE BSTRING HBYTE(4);
   DECLARE BSTRING LBYTE(4);
   DECLARE BSTRING NOP(4);

   LET WORD=B'000';
   LET HBYTE=B'001';
   LET LBYTE=B'010';
   LET NOP=B'111';

/* FUNCTION CODES */

   DECLARE BSTRING USERD(4); /* USER DATA */
   DECLARE BSTRING USERP(4); /* USER PROGRAM */
   DECLARE BSTRING SUPRD(4); /* SUPERVISOR DATA */
   DECLARE BSTRING SUPRP(4); /* SUPERVISOR PROGRAM */
   DECLARE BSTRING SUPRU(4); /* SUPERVISOR UNDEFINED */
   DECLARE BSTRING SUPRI(4); /* INTERRUPT ACKNOWLEDGE */

/* FUNCTION CODE VALUES */

   LET USERD=B'001';
   LET USERP=B'010';
   LET SUPRD=B'101';
   LET SUPRP=B'110';
   LET SUPRU=B'100';
   LET SUPRI=B'111';
```

```
/*************    STANDARD  READ    ********************/
 INSERT READS=4;
        IH(CLK,DTACK) ID(DATA) OI(FC,ADDR,RW);
        IL(CLK) OS(ADDR,FC,RW) OG(ADDR=?1/D'2',FC=?3) OH(RW);
        IH(CLK) OG(DS=?2);
        IL(CLK) IL(DTACK) IC(DATA) IG(DATA=?4);
        IH(CLK);
        IL(CLK);
        IH(CLK);
        IL(CLK) OH(DS);
 END INSERT;

 *******  READ TO INCLUDE TWO EXTRA CLOCK PULSES - RESYNC  *******/
 INSERT READSYNC=4;
        IH(CLK,DTACK) ID(DATA) OI(FC,ADDR,RW);
             IL(CLK)   /*  TWO EXTRA CLOCK PULSES  */ ,
             IH(CLK),
             IL(CLK),
             IH(CLK),
        IL(CLK) OS(ADDR,FC,RW) OG(ADDR=?1/D'2',FC=?3) OH(RW);
        IH(CLK) OG(DS=?2);
        IL(CLK) IL(DTACK) IC(DATA) IG(DATA=?4);
        IH(CLK);
        IL(CLK);
        IH(CLK);
        IL(CLK) OH(DS);
 ND INSERT;

 ***********     STANDARD  WRITE    **************/
 NSERT WRITES=4,
        IH(CLK,DTACK) ID(DATA) OI(FC,ADDR) OH(RW);
        IL(CLK) OS(ADDR,FC) OG(ADDR=?1/D'2',FC=?3);
        IH(CLK) OL(AS,RW);
        IL(CLK) OS(DATA) OG(DATA=?4);
        IH(CLK) IL(DTACK) OG(DS=?2);
        IL(CLK),
        IH(CLK);
        IL(CLK) OH(DS) OI(DATA);
 ND INSERT;
```

```
/************   CLOCK STATEMENT    ************/
.CLOCK CLK1;
      AT (CLK)
      F=2MEG
      NINC=8
      HISLEW
      DST(EDO,T2,S4)
      SIGNAL (H0,L2,H4,L6);
.END CLOCK;

/*************     SPEEDPLUS  READ       ****************/
.INSERT READ=3;
        TL(DS) OS(ADDR,FC,RW) OG(ADDR=?1/D'2',FC=?2) OH(RW);
        IL(DTACK) IC(DATA) IG(DATA=?3);
        TH(DS) IH(DTACK) ID(DATA) OI(FC,ADDR,RW);
.END INSERT;

/*************     SPEEDPLUS  WRITE    ****************/
.INSERT WRITE=3;
        TL(AS) OS(ADDR,RW,FC) OL(RW) OG(ADDR=?1/D'2',FC=?2);
        OS(DATA) OG(DATA=?3);
        TL(UDS,LDS) IL(DTACK);
        TH(DS) IH(DTACK) OI(DATA,RW,FC,ADDR);
.END INSERT;
```

Figure 6-15. Standard and SpeedPlus Testing of the MC68000 Microprocessor (page 1 of 3).

```
/********   INSERT TO INCREMENT THE PROGRAM COUNTER   **********/
.INSERT INCPC,
        LET PC=PC+D'2',
 END INSERT.
```

```
/*********    STANDARD RESET FOR FORCES     ***********/
.INSERT RESETS;
        IL(CLK) IH(BGACK,BERR,BR,RESET,HALT,DTACK,IPL0,IPL1,IPL2,VPA),
        IL(RESET,HALT);
                LOOP=D'512' $IH(CLK);
                $ IL(CLK) END LOOP;
        IH(RESET,HALT,CLK);
        LOOP=D'100' $IL(CLK);
        EXITIF PASSES IH(CLK) OS(DS,RW) OL(DS) OH(RW)  END LOOP,
.END INSERT;

/******   STANDARD - USED WITH INITIALIZATION WITH FORCES   *******/
.INSERT FINITIALS;
        .'RESETS';
        IL(CLK,DTACK) IG(DATA=X'0002');
        LOOP=D'2' IH(CLK);
        IL(CLK) END LOOP;
        LET PC=X'000002';
        .'FREADS'(X'0000');
        .'INCPC';
        .'FREADS'(X'0000');
        .'INCPC';
        .'FREADS'(X'C000');
        LET PC=X'00C000';
        .'FREADS'(X'4240');
        .'INCPC';
        .'READSYNC'(PC,WORD,SUPRP,X'4241');
        .'INCPC';
.END INSERT;

/*************  STANDARD READ USED WITH STANDARD FORCES  ****************/
 INSERT FREADS=1;
        IH(CLK,DTACK) ID(DATA);
        IL(CLK);
        IH(CLK);
        IL(CLK) IL(DTACK) IC(DATA) IG(DATA=?1);
        LOOP=D'2' IH(CLK);
        IL(CLK) END LOOP;
 END INSERT;
```

```
/********    STANDARD  DISABLES  AND  INHIBITS      *******/
 DISAB (ADDR,DATA,FC,DS,RW,VMA),
        IL(RESET) PL1(HALT);
        LOOP=D'10' $IL(CLK);
        $IH(CLK) END LOOP,
            HOLD(RESET,HALT);
 END DISAB;

 INH (ADDR,DATA,FC,DS,VMA,RW,BG,EN);
        IL(RESET) PL1(HALT);
        LOOP=D'10' $IL(CLK);
        $IH(CLK) END LOOP;
           .HOLD(RESET,HALT);
 END INH,
```

```
/**********     STANDARD   FORCE     SECTION     **********/
.LFORCE(LADDR);
        .'RESETS';
 END LFORCE.

.LFORCE(HADDR);
        .'RESETS';
.END LFORCE;

.HFORCE(LADDR);
        .'RESETS';
        IC(DATA) IL(CLK,DTACK) IG(DATA=X'0002');
        LOOP=D'2' IH(CLK);
        IL(CLK) END LOOP,
        LET PC=X'000002';
        .'FREADS'(X'0000');
        .'INCPC';
        .'FREADS'(X'00FF');
        .'INCPC';
        .'FREADS'(X'FFFF');
        .'INCPC';
        LET PC=X'FFFFFF';
        IH(CLK,DTACK) ID(DATA);
        IL(CLK);
        IH(CLK);
.END HFORCE;
```

```
/*********  SPEEDPLUS RESET MACRO FOR FORCES  ***********/
 INSERT RESET;
        IH(BGACK,BERR,BR,RESET,HALT,DTACK,IPL0,IPL1,IPL2,VPA);
        IL(RESET,HALT);
                FLOOP=D'712' ,
                END FLOOP,
        IH(RESET,HALT);
        TL(DS),
 END INSERT.

/*********  SPEEDPLUS -  USED FOR INITIALIZATION WITH  FORCES  *******
 INSERT FINITIAL;
        'RESET';
        IL(DTACK) IG(DATA=X'0002');
        TH(DS);
        LET PC=X'000002';
         'FREAD'(X'0000');
         'INCPC';
        .'FREAD'(X'0000');
         'INCPC';
        .'FREAD'(X'C000');
        LET PC=X'00C000';
        .'FREAD'(X'4240');
        .'INCPC';
        .'FREAD'(X'4241');
        .'INCPC';
 END INSERT;

/***************  SPEEDPLUS READ USED WITH FORCES  *****************/
.INSERT FREAD=1,
        TL(DS);
        IL(DTACK) IC(DATA) IG(DATA=?1);
        TH(DS) IH(DTACK) ID(DATA);
 END INSERT;
```

```
/********    SPEEDPLUS  DISABLES AND INHIBITS       ***************/
 DISAB (ADDR,DATA,FC,RW,VMA)
        USING(CLK1,F=ADDR DATA FC RW VMA RESET HALT);
   FAST,
        IL(RESET) PL1(HALT);
        FLOOP=D'100' ;
        END FLOOP,
   END FAST;
        .HOLD(RESET,HALT);
 END DISAB;

 INH (ADDR,DATA,FC,DS,VMA,RW,BG,EN)
      USING(CLK1,F=ADDR FC DS VMA RW BG EN RESET HALT);
   FAST,
        IL(RESET) PL1(HALT);
        FLOOP=D'100' ;
        END FLOOP,
   END FAST;
         HOLD(RESET,HALT);
 END INH;
```

```
/***************  SPEEDPLUS  H, LFORCE  SECTION  ******************/
 LFORCE(LADDR) USING(CLK1,F= ADDR INTC BGACK BR BG
                             BERR VPA RESET HALT ,T= DS EN);
   FAST;
        'RESET';
   END FAST,
 END LFORCE.

 LFORCE(HADDR) USING(CLK1,F= ADDR INTC BGACK BR BG
                             BERR VPA RESET HALT ,T= DS EN);
   FAST;
         'RESET';
   END FAST,
 END LFORCE.

 HFORCE(LADDR) USING(CLK1,F= ADDR DATA FC RW DTACK INTC BGACK BR BG
                             BERR VMA VPA RESET HALT ,T= DS EN);
   FAST,
         'RESET';
        IC(DATA) IL(DTACK) IG(DATA=X'0002');
        TH(DS);
        LET PC=X'000002';
         'FREAD'(X'0000');
         'INCPC';
        .'FREAD'(X'00FF');
         'INCPC';
         'FREAD'(X'FFFF');
        .'INCPC';
        LET PC=X'FFFFFF';
        IH(DTACK) ID(DATA);
        TL(DS);
   END FAST;
 END HFORCE.
```

Figure 6-15. Standard and SpeedPlus Testing of the MC68000 Microprocessor (page 2 of 3).

```
HFORCE(HADDR);
       .'RESETS';
       IC(DATA) IL(CLK,DTACK) IG(DATA=X'0002');
       LOOP=D'2 IH(CLK);
       IL(CLK) END LOOP;
       LET PC=X'000002';
        'FREADS'(X'0000');
        'INCPC';
        'FREADS'(X'00FF');
        'INCPC';
       .'FREADS'(X'FFFF');
       .'INCPC';
       LET PC=X'FFFFFF';
       IH(CLK,DTACK) ID(DATA);
       IL(CLK);
       IH(CLK);
END HFORCE;
```

```
       /*********************************************************/
       /************  STANDARD   MAIN  PROGRAM   ****************/
       /*********************************************************/

MAIN,

BURST;

 $PU(HALT);

/*  SET CONTROL LINES(INPUTS) TO INACTIVE STATE  */
       IC(CLK,VPA,BERR,RESET,HALT,IPL0,IPL1,IPL2,BGACK,BR,DTACK)
       IL(CLK)
       IH(VPA,BERR,RESET,HALT,IPL0,IPL1,IPL2,BGACK,BR,DTACK);

/*  LOOP TO GET ENABLE LINE (PIN 20) TO A KNOWN STATE  */
       LOOP=D'256' $IH(CLK);
       EXITIF PASSES IL(CLK) OS(EN) OH(EN) END LOOP,
         OI(EN);

/*   TOTAL RESET OF MICROPROCESSOR  */
       IL(RESET,HALT);

/* CLOCK PROCESSOR TO ENABLE RESET,HALT  */
       LOOP=D'800' $IH(CLK);
       $IL(CLK) END LOOP;

/* HALT AND RESET HIGH , READY TO START INITALIZATION  */
       IH(RESET,HALT,CLK);

       OS(BG) OH(BG);

/* CLOCK PROCESSOR TILL ADDR $0000 AND DATA/ADDR STROBES COME UP */
       LOOP=D'100' $IL(CLK);
       EXITIF PASSES IH(CLK) OS(ADDR,DS,RW,EN,VMA)
                             OL(ADDR,DS) OH(EN,VMA,RW)  END LOOP;
        OI(EN);

/*  THESE NEXT 4 READS ARE REQUIRED TO SETUP THE MC68000 INTERNAL STACK  */
/*  AND PROGRAM COUNTER AFTER A RESET                                    */

/* LOAD $0002 -> SUPERVISOR STACK POINTER - HIGH */
       IC(DATA) IL(CLK,DTACK) IG(DATA=X'0002');
       IH(CLK);
       IL(CLK),
       IH(CLK);
       IL(CLK) OH(DS);

/*   INITIALIZE PROGRAM COUNTER  */
       LET PC=X'000002';

/* LOAD $0000 -> SUPERVISOR STACK POINTER - LOW  */
        'READS'(PC,WORD,SUPRP,X'0000'),
        'INCPC';

/* LOAD $00FF -> PCH */
        'READS'(PC,WORD,SUPRP,X'00FF');
        'INCPC';

/* LOAD $2000 -> PCL */
        'READS'(PC,WORD,SUPRP,X'C000');
        'INCPC';

/*  RESET PROGRAM COUNTER TO FFC000  */
       LET PC=X'FFC000';

/*  F + E 4E71 (NOP) --  NO OPERATION OCCURS  */
       IH(CLK,DTACK) ID(DATA) OI(ADDR,DS,RW),
       IL(CLK) OS(ADDR,FC,RW) OG(ADDR=PC/D'2',FC=SUPRP) OH(RW),
       IH(CLK) OS(DS) OL(DS),
       IL(CLK) IL(DTACK) IC(DATA) IG(DATA=X'4E71'),
       IH(CLK);
       IL(CLK,VPA);  /* ASSERT VPA TO EXERCISE  M6800 INTERPACE PINS  */
       IH(CLK),
       IL(CLK) OH(DS),
        'INCPC',
```

```
.HFORCE(HADDR) USING(CLK1,F= ADDR DATA FC RW DTACK INTC BGACK BR BG
                             BERR VMA VPA RESET HALT ,T= DS EN);
  FAST,
       .'RESET';
       IC(DATA) IL(DTACK) IG(DATA=X'0002');
       TH(DS);
       LET PC=X'000002';
       .'FREAD'(X'0000');
       .'INCPC';
       .'FREAD'(X'00FF');
       .'INCPC';
       .'FREAD'(X'FFFF');
       .'INCPC';
       LET PC=X'FFFFFF';
       IH(DTACK) ID(DATA);
       TL(DS);
  END FAST,
END HFORCE,
```

```
       /*********************************************************/
       /**********   SPEEDPLUS MAIN  PROGRAM  *****************/
       /*********************************************************/

 MAIN USING(CLK1,F= ADDR DATA FC RW DTACK INTC BGACK BR BG BERR VMA VPA
                    RESET HALT ,T= DS EN);

BURST VTT=1.45;

 $PU(HALT);

FAST,

/*  SET CONTROL LINES(INPUTS) TO INACTIVE STATE  */
       IC(VPA,BERR,RESET,HALT,IPL0,IPL1,IPL2,BGACK,BR,DTACK)
       IH(VPA,BERR,RESET,HALT,IPL0,IPL1,IPL2,BGACK,BR,DTACK),

/*  TOTAL RESET OF MICROPROCESSOR  */
       TH(EN);

       IL(RESET) PL(HALT);

/*  CLOCK PROCESSOR TO ENABLE RESET,HALT  */
       FLOOP=D'2000' ;
       END FLOOP,

       IH(RESET,HALT);

       TL(DS) OS(BG) OH(BG),
       IL(DTACK);
/*  4 READS ARE REQUIRED TO SETUP THE 68000 INTERNAL STACK  */
/*  AND PROGRAM COUNTER AFTER A RESET                       */

/* LOAD $0002 -> SUPERVISOR STACK POINTER - MSByte */
       IC(DATA) IG(DATA=X'0002') IL(DTACK);
       TH(DS) IH(DTACK);


/*   INITIALIZE PROGRAM COUNTER  */
       LET PC=X'000002';

/* LOAD $0000 -> SUPERVISOR STACK POINTER - LSByte  */
       .'READ'(PC,SUPRP,X'0000');
       .'INCPC';

/* LOAD $00FF -> PC-MSword */
       .'READ'(PC,SUPRP,X'00FF');
       .'INCPC';

/* LOAD $2000 -> PC-LSword */
       .'READ'(PC,SUPRP,X'C000');
       .'INCPC';

/*  RESET PROGRAM COUNTER TO FF2000  */
       LET PC=X'FFC000',

/*  F + E 4E71 (NOP) --  NO OPERATION OCCURS  */
       IH(DTACK) ID(DATA) OI(ADDR,RW),
       TL(DS) OS(ADDR,FC,RW) OG(ADDR=PC/D'2',FC=SUPRP) OH(RW),
       IC(DATA) IG(DATA=X'4E71'),
       IL(DTACK);
       TH(DS) IH(DTACK) ID(DATA) OI(FC,RW,ADDR) OS(VMA) OH(VMA),
       .'INCPC',
```

Figure 6-15. Standard and SpeedPlus Testing of the MC68000 Microprocessor (page 3 of 3).

Clearly, ED can also be used to synchronize a series of test steps to the timing cycle. For example, a component with complex timing requirements may need an alternating pattern of drive-sense timings, for example,

DST(T1,D2,S3, ED6, S10),

which senses immediately after the first drive strobe, but waits 4 increments before sensing after second drive strobe (ED6). It will be necessary to align the test steps with these alternating timings; this can be accomplished using a trigger nail tied to some constant voltage level, such as ground, and starting the test sequence with a statement that triggers on this voltage.

TL(GND) IC(<IC-pins>) OS(<IC-pins>);

At T1 the ground nail is sampled low, of course, but the statement cannot execute until ED occurs. Hence, it is known that this first step executes with the timing D6, S10; the next step executes on D2, S3, and the following step returns to D6, S10.

The hardware requires a minimum of 25 ns between the end of a drive strobe and the beginning of a sense strobe. Therefore, when the frequency is high, TINC could become too small to have drive and sense strobes in consecutive increments.

Speed in SpeedPlus Testing

Users of SpeedPlus tests must bear in mind that the purpose of the SpeedPlus in-circuit test subsystem is to provide efficient functional tests of components that require complex test timing or high minimum test rates. It is not meant for testing component timing parameters such as gate delays or maximum device speed, which are usually very difficult to measure in the in-circuit test environment. A test should always use the lowest test rate possible because, as the test rate increases, the test becomes more sensitive to timing delays and waveform inaccuracies, and it becomes harder to debug. Bear in mind also the difference between clock rate and test rate; clock frequencies can be programmed to 20 MHz, but test steps can be executed at no more than 5 MHz -- that is, 5 million test vectors per second. High clock frequencies pose less of a problem in test development than high test rates.

When a SpeedPlus test must run near the 5-MHz maximum test rate (e.g., when it must synchronize to an on-board clock that cannot be disabled), special measures will probably be needed to allow for the timing tolerances imposed by the test system, the fixture, and the component's circuit board environment. One potential problem arises from input sensing (the default IS term implied in every input connect -- IC). All driven logic levels are normally sensed by the driver/sensor hardware to ensure that the programmed logic levels are in fact being applied to the UUT (i.e., to ensure that the UUT input is not shorted). To meet the set-up times of the sensors, however, it may be necessary to lower their sense thresholds (VIHA, VILB, etc.) so that a driver's logic level can be latched in correctly before the driver has finished slewing to its programmed state. In some case it may be desirable to disable input sensing with the term II (input ignore).

In cases where a low impedance node must be driven, it will be found that the driver slews more slowly than normal and that its timing may need to be advanced to compensate. This can be done by specifying an earlier drive time in the DST= parameter of the SET CLOCK statement, or by moving the IH and IL terms

to points one or more test steps earlier than would otherwise be required. For further information about driver sensor timing, refer to Test Programming, para 4.4. Special attention must also be given to the kinds of connections and terminations used in the test fixture; fixturing advice for high-speed testing appears in the Test Fixture Kit Instruction Manual for your system.

6.3 DIGITAL TEST STRUCTURE LANGUAGE (DTSL)

The Digital Test Structure Language can be regarded roughly as a superset of the digital test language described in the Test Programming manual. However, the test language statements are used in DTSL in somewhat different ways than they are used in test programs. (For example, IC pin numbers are specified in DTSL rather than test nail numbers.) Additional statements allow pin states to be specified conditionally, depending on the circuit environment that DTG finds when it uses the DTSL procedure to write a test. It may be helpful to regard DTG as a kind of macro processor that expands procedures written in a macro language (DTSL) into statements in the 227X digital test language. The statements unique to DTSL are defined in paragraphs 6.3.1 through 6.3.3; a list of all DTSL keywords (including those taken from the test language) appears in Table 6-1.

For definitions of the commands shared with the digital test language (e.g., IH(), OS(), etc), refer to the Test Programming manual, para 6.4. Similar commands unique to DTSL which allow conditional pin specifications are defined in para 6.3.3 (e.g., PH(), NP(), OV()).

Although DTSL provides constructs similar to those in the digital test language, some important differences should be noted. The only allowed variables are bit strings, which can be from 1 to 32 bits long. Except in the HEADer section, unquoted numbers stand for pins, not numerical values. Thus, if ONE is declared as a BSTRING (bit string) variable, LET ONE=2 means, assign the current state of pin 2 (high or low) to the variable ONE. Numbers, or numerical constants, can only be represented by digits placed in quotes and preceded by a letter that gives the base, for example, LET ONE=B'1'. This assigns the binary digit 1 to the variable ONE. A LOOP statement must use this representation, e.g., LOOP=D'8'. (INSERT definitions, however, may use unquoted digits to specify the number of parameters, e.g., INSERT name=2 specifies 2 parameters.) The statements SET LGC and TEST LGC are recognized as in the test language (except that they must specify device pins rather than test nails), and TEST LGC is the default statement type. The $ may be used for SET LGC but does not require a following space.

Like the test language, DTSL preserves pin states from one statement to the next. However, conditional statements like CLUSTer and IF-THEN-ELSE restore the pin states, or context, that exists before them.

An underline character (_) in a language format or keyword indicates that a space is required.

6.3.1 Words and Symbols

DTSL Keywords

Section-Defining Keywords

.CLUST	.HEAD	.HFORCE	.SPEC
.DISAB	.INH	.LFORCE	.SUB
.END	.INSERT	.MAIN	USING

DTSL Keywords (cont)

IC Definition Keywords

DH	DU	OPENE	TRI
DISJ	INPUT	OUTPUT	UNDRIV
DL	OPENC	SIZE	

Command Keywords

.CHOOSE_C	.HOLD	.ZEROCONTEXT
.CHOOSE_I	.LET	.ZEROSTATE
.CONTEXT	.SAY	

Pin Function Keywords

CONN	IG	OG	SETIND
DEF	INDEP	OV	TG
FREE	KNOWN	PH	UNDEF
GND	NP	PL	VCC

Flow Control Keywords

.ELSE	.END_REPEAT	.IF
.ELSE_IF	.FI	.REPEAT

DTSL Operators

Arithmetic operators

+	(add)	*	(multiply)
-	(subtract)	/	(divide)

Logic operators

OR	XOR	NAND
AND	NOT	NOR

General Definitions

pin-list Many of the DTSL statements reference one or more IC pins. This reference is made through specific types of pin specifications as follows:

1. pin-designation
2. group-name
3. equivalent-set-name

Any combination of one or more of these pin specifications separated by commas makes a pin-list. An arithmetic or a logical expression may be used to derive any of these pin specifications.

pin-designation The IC pin number which is an integer from 1 through 480.

group-name Name of a group of pins whose state is defined by corresponding values of a bit string. The group must be defined in a DECLARE GROUP statement. Group names may be used in any appropriate library command. DTG will replace the name with the actual pin list and pin states. The description of the keyword OG includes a complete description of the use of groups. Long group names will be truncated to 8

group-name (cont) — characters by the library. A group may contain no more than 32 pins.

equivalent-set — Two or more pin sets separated by commas where each pin set consists of pin-designations separated by spaces. Example: (1 4 5 9, 2 5 6 10).

equivalent-set-name — A name given to a set of IC pins which provide identical functions in independent sections of the IC. Equivalent-sets are used in HEADer section statements. Long equivalent-set names will be truncated to 8 characters by the library.

expression — An expression is a combination of constants, variables and system-defined functions that are operated upon by arithmetic and logical operators. The result of evaluating the expression is a 32-bit integer value. This value is interpreted according to the context of the statement in which the expression occurred. The interpretation may be a pin group, a BSTRING, the count in a REPEAT, or a logical condition indicator. The logical condition indicator may be set to 0 or its 1s complement, -1. A logical decision statement will interpret a 0 as a false condition and -1 as a true condition. Any other non-zero indicator referenced in a case construct will also be treated as a true condition.

Bit string values are determined by evaluating the integer expressed in the base 'number' notation, where base is one of the number system indicators:

B = binary	D = decimal
O = octal	X = hexidecimal

The range of numbers allowed in each system is any number that can be represented with 32 binary bits, as follows:

B	0 - 32 1s	D	0 - 8589934591
O	0 - 37777777777	X	0 - FFFFFFFF

System-Defined Functions

These functions will provide condition indicators for each item in the function expression. When any of the item indicators are zero, then the expression is considered false. When all indicators are 1, then the expression is true. The system functions are recognized by the following keywords:

Function Keyword	Condition
CONN	Pins connected
DEF	Item defined
FREE	Pins not connected to VCC or GND
GND	Pins connected to GND
INDEP	Pins not connected to each other
KNOWN	Pins state known
SETIND	No connections exist between pin sets
UNDEF	Item not defined
VCC	Pin connected to VCC

See the definitions of keywords for full descriptions.

Function Keyword (Cont'd)	Condition
KNOWN	Pins state known
SETIND	No connections exist between pin sets
UNDEF	Item not defined
VCC	Pin connected to VCC

See the definitions of keywords for full descriptions.

Examples of DTSL Statements

```
$IG(ADDR=ADDR*D'2');
```

Meaning: The pin states of the input group ADDR will be changed from the original states of ADDR to the states resulting from the multiplication of the original ADDR by the decimal number 2. If the original ADDR bit string was 000001 and this instruction was in a REPEAT construct with 6 repetitions, each of the pins represented by the 6 bits of ADDR would successively have the value of 1. The 6 input pins would have been defined in a DECLARE GROUP statement. The expression is evaluated and the resulting bit string value is interpreted into highs (1) and lows (0) for the pins in the group ADDR.

```
OS(Y,W) OV(Y=DA, W=.NOT.DA);
```

Meaning: Sense the outputs Y and W. The expected value of Y is the current value of pin DA and the expected value of W is the inverse value of DA. The expressions are evaluated by checking the context of pin DA and applying the results to Y and W.

```
.IF(CONN(J,NOTQ).OR.CONN(K,NOTQ).AND.INDEP(J,K));
```

Meaning: If pins J and NOTQ are connected together or if pins K and NOTQ are connected together and pins J and K are independent, then perform the next statement. Otherwise, go to the next ELSE IF, ELSE, or FI. Each of the 3 functions is evaluated and provides an indicator to show if the condition exists. Then the logical operators are evaluated so that if either CONN indicator is 1 and if the INDEP indicator is 1, then the IF conditional is considered true. Logical expressions are evaluated in the order .NOT., .AND., .OR.

```
OG(OUT=OUT+OUT+RIN);
```

Meaning: The pin states of the output group OUT will be changed from their current state to a state that is equal to the binary addition of their current states plus their current states plus the current states of group RIN. If the current states of OUT = 00001 and the current states of RIN = 00100, then the new states of OUT are 00110, as calculated below.

```
  00001     00010
+ 00001   + 00100
-------   -------
  00010     00110
```

```
LET ADDR = X'01';
```

Meaning: The context of the pins in the group ADDR is set to correspond to the binary bits of the hexidecimal number 01.

6.3.2 Language Structures

Sections

A procedure written in DTSL is so structured that all statements, except the DECLARE and LET statements, must be contained within one of the section structures. Every test procedure must start with a HEADer section and must contain the main flow of the burst within a MAIN section. All other sections are optional and must appear between the HEADer and MAIN sections. If an IC must be tested using different procedures, depending upon how it is connected in the circuit, one or more SPECial sections may be used to provide the burst procedure. DTG will decide whether to use the SPECial or to use the MAIN, based on the logical condition specified in the SPECial section. When more than 1 SPECial section appears, DTG uses the first one whose selection condition is met and discards all others (including MAIN). Sections meant to run on the optional SpeedPlus hardware are marked by the term USING followed by a list containing a reference to a clock specification that defines the desired timing and a list of the component pins requiring SpeedPlus nails.

DISABle and INHibit sections are used to isolate ICs being tested from the effects of other ICs that may be on a common bus, or in a feedback loop, or in a free-running clock circuit. HFORCE and LFORCE sections identify the bused output pins to be forced to high or low states. The xFORCE sections are used within BUSTEST blocks to locate any stuck-high or stuck-low output pins. (See Figures 6-1 and 6-2 for an overview of how these sections contribute to the test programs DTG writes.) INSERT and SUBroutine sections are used to define modules of a test procedure that are written outside the MAIN or SPECial sections. They can be called from within any subsequent sections. An INSERT is inserted into the burst procedure at DTG time. Program control jumps to a SUBroutine at run time.

The order in which these optional sections appear in the test procedure is arbitrary except that any defining sections must precede the procedure section in which it is referenced. That is, if an INSERT is used in a SPECial section, the INSERT must appear prior to the SPECial in which it is used. Any section that requires resources from the optional SpeedPlus subsystem is identified by a USING clause following the section keyword.

A summary of the sections in a DTSL model is:

Required	Optional
HEADer	DISABle
MAIN	INHibit
	INSERT
	HFORCE
	LFORCE
	SPECial
	SUBroutine

Section Format

```
.section-ID null/argument;
    ¦
```

```
    DTSL and test language statements
        ¦
        ¦
.END section-ID;
```

where: section-ID Section keyword (HEAD,MAIN,INSERT,SUB,DISAB,SPEC, INH,HFORCE,LFORCE)

argument Name/pin-list/conditional

Formats for specific groupings are included in para 6.3.3.

IF-THEN-ELSE (Case) Construct

This construct is used to make decisions based on certain conditions. These conditions may be expanded into a 'case' construct with any number of conditional selection expressions.

If the result of evaluating the expression is true, the statement following the IF (condition) or the ELSE IF (conditional) is executed. Otherwise, the next ELSE IF (conditional) is evaluated or the statement after the ELSE or the FI is executed. These choices are based strictly on which is next in the construct. Each construct must start with an .IF (conditional); and end with a .FI;. An .ELSE IF (conditional) and an .ELSE are optional in the construct. There may be any number of .ELSE IF (conditional) statements. There can only be one .ELSE, which must precede the last case.

The result of a case construct will be the inclusion of selected test language statements (case) into the generated test procedure based on the decisions made in the construct. The context of pin states following the case construct is the same as the context just prior to the case construct.

IF-THEN-ELSE (Case) Construct Format

```
.IF conditional;
    ¦
  test language statements for case 1
    ¦
    ¦
.ELSE IF conditional;
    ¦
  test language statements for case 2
    ¦
    ¦
.ELSE;
    ¦
  test language statements for last case
    ¦
    ¦
.FI;
```

where: IF and FI . Required construct keywords

ELSE IF and ELSE Optional construct keywords

conditional Any expression that can be evaluated by the system

expression Refer to description in General Definitions

6.3.3 Definitions of DTSL Keywords and Statements

The DTSL command keywords and function keywords are defined in the following paragraphs. They are arranged in alphabetical order of keywords.

--------Call INSERT--------

General

A string of statements defined as an INSERT may be called for insertion into the test procedure in one of two ways. One is a selective call via a CHOOSE I command. The other is an absolute call which consists of the name of the insert plus an argument list if arguments are needed. This paragraph defines the absolute call.

Format

1. .'insert-name'(arg-list),

2. .'insert-name';

where: insert-name — Character string that identifies the insert

arg-list — arg1, arg2, ...

arg — Character string that is any part of a DTSL statement (if a partial pin-list is used as the argument, the argument must be enclosed in quotes (")).

Examples

1. .'WRITE'(ZERO, IH);

   Meaning: Replace this command call with the insert whose name is WRITE. The first argument of the insert is the variable ZERO and the second argument is the pin state IH. The value of ZERO will replace the symbols ?1 and IH will replace the symbols ?2 in the insert.

2. .'FOUR';

   Meaning: Replace this call statement with the insert named FOUR.

Comments

1. The maximum number of arguments allowed in an argument list is 9.

2. Commas are recognized as separators between arguments so they are not allowed in an argument except where parentheses are used, as in a pin list.

3. Quotes may be used to enclose an argument that contains a comma, such as an incomplete pin list.

4. Inserts may be nested.

5. An insert becomes part of the main program flow.

6. The context at the beginning of an insert is that of the step just prior to the place where the insert is inserted.

7. The context of the step following the selected insert is that of the last step in the insert.

8. Long insert names will be truncated to 8 characters by the library.

---------CHOOSE Call---------

General

Instructs DTG to select a subroutine from the list provided in the statement. DTG will analyze each subroutine in the order listed and will insert a subroutine call statement (GOSUB name) into the test program. This calls the first acceptable subroutine.

Format

```
.CHOOSE C(subroutine-list);
```

where: CHOOSE C — Command keyword

subroutine-list — Subroutine-name1, subroutine name2,...

subroutine-name — Character string

Example

```
.CHOOSE C(EXSUB);
      |
      |
.SUB EXSUB
      |
      |
subroutine statements
      |
      |
.END SUB;
```

Meaning: The statements that are in the subroutine EXSUB will be executed in place of the .CHOOSE C statement.

Comments

1. A subroutine call is inserted only if a subroutine can be chosen that is fully executable without any deletions.

2. The subroutine remains separate from the main program.

3. The context at the beginning of the subroutine is zero.

4. The context of the first step after the subroutine call is the same as that of the step just prior to the subroutine call.

5. Long subroutine names will be truncated to 8 characters by the library.

---------CHOOSE Insert----------

General

A string of statements defined as an INSERT may be called for insertion into the test procedure in one of two ways: an absolute call (.'insert-name') or a selective call via a CHOOSE I statement. This paragraph defines the selective call. Either of the calls may, or may not, include arguments. CHOOSE I instructs DTG to evaluate inserts in the list and use the first one that can be executed.

Format

.CHOOSE I(insert-list);

where:		
	CHOOSE I	Command keyword
	insert-list	Insert-entry, insert-entry,...
	insert-entry	Insert-name or insert-name(arg-list)
	insert-name	Character string that identifies the insert
	arg-list	arg1, arg2, ...
	arg	Character string that is any part of a DTSL statement (if a partial pin-list is used as the argument, the argument must be enclosed in quotes).

Examples

1. .CHOOSE I(CLRA, CLRB);

   Meaning: Evaluate insert CLRA. If it is fully executable, insert it in place of the CHOOSE I statement. Otherwise, evaluate insert CLRB. If it is fully executable, insert it in place of the CHOOSE I statement. Otherwise, no insert is made.

2. .CHOOSE I(INSA("IH(1,2,3)","OH(4)"), INSB("OH(4)"),INSC);

   Meaning: Evaluate insert INSA which has the two arguments "IH(1,2,3)" and "OH(4)". If it is not usable, evaluate insert INSB which has an argument "OH(4)". If INSB is also not usable, evaluate INSC which has no arguments.

Comments

1. An insert is acceptable only if every one of its steps is executable without any deletions. However, if a lower level of embedded cluster is deleted, the insert is not deleted.

2. An insert becomes part of the main program flow.

3. The context at the beginning of an insert is that of the step just prior to the place where the insert is inserted.

4. The context of the step following the selected insert is that of the last step in the insert.

----------CLOCK----------

General

On systems equipped with the SpeedPlus option, sections of the library procedure can be marked with a .USING term in the .HEADER section to produce tests meant to run on the SpeedPlus subsystem. Timing for these sections can be defined either by a implicit, default clock statement or by a named, explicit clock statement. If the term .FREQ or .PERIOD appears in the .HEADer section, an implicit clock is defined within the specified frequency range and the standard defaults. Otherwise, at least one .CLOCK statement must appear in the procedure.

When DTG writes a burst into the .DTP file, it must precede it with a single SET CLOCK statement that meets the requirements of all disables, inhibits, and bus tests (forces) in the burst, as well as the requirements of the UUT. The actual clock parameters chosen for the SET CLOCK statement are based on limits derived from all of the library procedures contributing to the burst. If conflicting (non-overlapping) requirements are made by the various sections, the main section has precedence and other sections are discarded.

If a SET CLOCK statement is defined in an ATO file associated with the circuit description (.CKT), the ATO clock will override all clocks defined in the library procedures. Refer to Test Set Preparation, para 5.2, for use of clock statements in .ATO files. Refer to the Test Programming manual, para 6.4, for a complete clock statement definition. DTG writes the SET CLOCK statement in accord with the following precedence for clock statement parameters: .ATX file; VALUE section of .CKT file; .CLOCK or .FREQ statement.

Format

```
.CLOCK clock-name;
  AT (CDA=pina : CDB=pinb : CDC=pinc : CDD=pind)
F=freq
PERIOD=prd
TINC=incrtime
NINC=count
SIGNAL (CDA=transition-list : CDB=transition-list : CDC=transition-list :
     CDD=transition-list : CSYNC=transition-list)
DST (D/S-timing-list)
VCDL=volts
VCDH=volts
VCST=volts
HISLEW ¦ LOSLEW
ON ¦ OFF;
.END CLOCK;
```

where:

clock-name	An arbitrary name for the clock statement.
pina, pinb, pinc, pind	Specifies pins to be driven by clock signals CDA to CDD. The terms CDA to CDD may appear in any order, separated by colons (:). Each clock driver is assigned to the named IC pin. Defaults: If only pin names are given, they are assigned in order to CDA, CDB, CDC, and CDD.

F=freq — The frequency at which the timing matrix is read out. Clock and drive/sense frequencies default to this value, but they may also be multiples of it if more than one drive/sense pair or more than one clock cycle is defined in the timing matrix (i.e., in "transition-list" or in "D/S-timing-list"). Equal to 1/prd. May not be used if PERIOD=prd is used; however, either F or PERIOD must be defined. The range of frequencies that can be specified is 20 kHz to 20 MHz.

PERIOD=prd — The basic period in which the timing matrix is read out. Equal to 1/freq. May not be used if F=freq is used; however, either PERIOD or F must be defined. The range of times that can be specified is 25 ns to 51.2 us.

TINC=incrtime — Defines the length in seconds of each increment of the timing matrix. incrtime * count = prd = 1/freq. May not be used if both "count" and either "prd" or "freq" are already defined.
Default: PERIOD/2 or 2/FREQ (when not defined by PERIOD or F and NINC). The range of times that can be specified is 50 ns to 51.2 us.

NINC=count — Defines the number of time increments in the timing matrix. count * incrtime = prd = 1/freq. Must be defined and must agree with the value implied by incrtime and prd.
Default: 4 (when not defined by PERIOD or F and TINC.) The range of number of increments that can be specified is 1 to 256.

SIGNAL — Contains the transition lists that define the clock signals.

CDx=transition-list
Each clock drive signal used can be specified by a transition list. A transition list contains up to 256 transitions, separated by commas. A transition consists of the letter H (for low-to-high) or L (for high-to-low) followed by the increment number in which the level transition takes place. The default transitions are NINC/2 when NINC is even and (NINC-1)/2 when NINC is odd; in other words, the default waveform is a square wave with a low-going edge at the first increment and a high-going edge half-way through the timing cycle. Triggers and Drive Enables are also included in the transition list.
Default: If a transition list is not supplied, the default is a signal with a 50% duty cycle, going high at t=PERIOD/2. When NINC is odd, the high transition is at t=(PERIOD-NINC)/2. Refer to TRIGGER HI/LO in HEADer.

DST=D/S-timing-list
Describes the times at which test events occur. The D/S-timing-list contains parameters Dt, St, Tt, and EDt, where t is a number in the range 0 to NINC - 1.

D names the time(s) at which drivers change states. In addition, each D calls out a new test step, except after test statements containing trigger terms. However, each D defaults to ED if no ED is given in the list.

S names the time(s) at which sensor states are sampled. Each S calls out a new test step except after statements containing a trigger.

T names the time(s) at which trigger nails are sampled.

ED names the drive time(s) at which test statements containing trigger conditions may execute.

Defaults:
- D - every transition in the lowest lettered CDx= parameter.
- S - the smallest interval allowed by the hardware, ranging from 50 to 200 ns before the drive strobe (D).
- T - the smallest interval allowed by the hardware, ranging from 60 to 200 ns, before the drive strobe (D).
- ED - if no ED is specified in the list, every ED is implied at every D.

VCDL=volts
VCDH=volts
VCST=volts

For VCDL and VCDH, specifies the logic drive voltages; for VCST specifies the low-to-high transition voltage at the clock module's synchronization input; range is -6 to +12 volts.

Defaults:	VCDL	VCDH	VCST
Logic Family			
A	0.2	3.5	1.5
B	-1.7	-0.8	-1.3

Comment

In general, .CLOCK statements should be avoided where DTG may need to resolve conflicting clock requirements of more than one component. Instead, a SET CLOCK statement giving the required parameters should be placed in the .ATO file associated with the circuit description to replace the DTG clock. Refer to Test Set Preparation manual, para 5.2.

----------CLUSTer----------

General

Circuit conditions may make it impossible to achieve some pin states specified in the procedure. In this case it is desirable to delete the group of statements that contain unachievable commands. Where this condition could exist the CLUSTer can be used to cause DTG to analyze the circuit and determine if all the statements in the CLUSTer can be performed. If not, the entire CLUSTer is deleted from the procedure.

Format

```
.CLUST;
   |
   |
  DTSL and test language statements
   |
   |
.END CLUST;
```

where: CLUST Group keyword

END CLUST Group keyword

Examples

1.
```
.CLUST;
     IC(A1) IL(A1,CLK,WDSEL);
     IH(CLK);
     IL(CLK);
     ID(A1);
     OS(QA) OL(QA);
.END CLUST;
```

Meaning: All the test statements in the CLUSTer are analyzed by DTG to see if the circuit configuration will allow all the specified test conditions to exist. If it will, the CLUSTer, minus the .CLUST; and .END CLUST; statements, is included in the generated test procedure.

2.
```
.CLUST;
    IL(A,B,C);
    PL(DATA) OS(Y,W) OV(Y=DA, W=.NOT.DA);
    .CLUST;
        PL(DA) PH(DB,DC,DD,DE,DF,DG,DH) OS(Y,W)
        OV(Y=DA,W=.NOT.DA);
    .END CLUST;
    .CLUST;
        PH(DA) PL(DB,DC,DD,DE,DF,DG,DH) OS(Y,W)
        OV(Y=DA,W=.NOT.DA);
    .END CLUST;
.END CLUST;
```

Meaning: The analysis of the highest level CLUSTer and the two nested CLUSTers will take place. If one of the nested CLUSTers is not executable, it will not be included in the test procedure with no affect on the remaining statements. If either of the first two statements, which are part of the higher level CLUSTer only, cannot be executed, the entire group of 3 CLUSTers is not written into the procedure.

Comments

1. If any statement in the CLUSTer is not executable, the entire CLUSTer is deleted.

2. A CLUSTer may be nested within another CLUSTer.

3. If a nested CLUSTer is deleted, the higher level CLUSTer is not deleted. Lower level CLUSTers are deleted.

4. The pin state context at the statement following a CLUSTer is that which existed at the end of the statement which preceded the CLUSTer. The context of the pin states in the CLUSTer is not retained when returning to the main program flow.

----------CONNected----------

General

This is a function which causes DTG to evaluate the circuit conditions for the pins in the pin-list to see if they are connected together. If any pins in the pin-list are connected together, DTG will return a true indication to the statement containing the CONN function.

Format

CONN(pin-list)

where: CONN Function keyword

pin-list Previously defined in General Definitions

Examples

1. .SPEC CONN(B,C);

   Meaning: If pin B and pin C are CONNected, a true status will be set and the SPECial group will be written instead of the MAIN group. Pins named B and C were assigned in the INPUT statement of the HEADer group.

2. .IF CONN(J,NOTQ).AND.CONN(K,Q);

   Meaning: If the NOTQ output is CONNected to the J input and if the Q output is CONNected to the K input of the J-K flip-flop, then both CONN conditions will be true and the commands that follow this statement will be performed.

3. .IF CONN(OUT,IN);

   Meaning: Output group OUT consists of pins 4 and 5; input group IN consists of pins 1, 2, and 3. If output pin 4 or 5 is connected to any input pin, 1-3, perform the next statement.

----------CONTEXT----------

General

Instructs DTG to insert pin states, as they would exist just prior to the statement with the indicated label, into the next statement. If no pin-list is included in the CONTEXT statement, all stated and implied pin states of the referenced statement are considered. If a pin-list is included, the stated and implied pin states prior to the referenced statement are considered and modified

by that pin-list. Pin states explicitly stated in the next statement are not affected by the CONTEXT statement. Implied pin states are those which were set in a statement which preceded the referenced statement. This is valid because of the incremental setting of pin states. The referenced statement must precede the CONTEXT statement. This is because DTG processes statements in order and has no way of looking ahead in the program. The referenced statement must have the statement label that is used in the CONTEXT statement. Pin states are permanently changed by the CONTEXT statement. The context for the step following the CONTEXT statement will be as determined by the CONTEXT statement.

Format

```
.CONTEXT statement-label:pin-list;
```

where: CONTEXT — Command keyword

statement-label — User-provided name with 1 to 8 ASCII characters. The first character must be a letter.

pin-list — Refer to General Definitions

Example

```
    IH(A);
    IL(A);
A3: IH(A);
      ¦
      ¦
    OV(OUT=A3:A);
```

Meaning: The state of the OUT pin is set high.

----------DEFined----------

General

A function is provided to allow for the checking of an element to see if it has been defined. This function is primarily for use in INSERTs to determine if an argument has been DEFined in the calling statement. The function is legal, however, for any reasonable application. DTG will return a true indication when any of the items in the DEFine list are DEFined.

Format

```
DEF(def-list)
```

where: DEF — Function keyword

def-list — One or more conditions separated with commas

Example

```
.INSERT SMPL=2;
    ¦
    ¦
  .IF DEF(?2);
    ¦
    ¦
  .FI;
    ¦
    ¦
.END INSERT;
```

Meaning: If the statement that currently has called this INSERT has DEFined the second argument, then perform the next statement. Otherwise, perform the statement after .FI.

---------DH (default high)---------

General

When DTG finds a situation at an IC input pin where either of the following conditions exists, it will not normally produce a test.

No driver/sensor assigned or VCC or GND not indicated

and

Pin state specified in library

In some cases it may be acceptable to generate a test when inputs are not connected. In order for this to happen, a default pin state must be specified in the HEADer section. This default state will be used any time the pin is not connected in the circuit description. This feature is especially useful when dealing with ECL devices where the ECL internal pull-down resistors allow low level inputs to be achieved by an open input.

The default state is specified as high by use of the DH command defined below.

Format

.DH(pin-list);

where: DH Command keyword

pin-list Previously defined in General Definitions

Example

.DH(1,6,10);

Meaning: Any time that pins 1, 6, and/or 10 are not connected in the circuit description, default to the high state and produce a test.

----------DISABle----------

General

Indicates the start of a section that describes how to disable bus-type output pins. The output-pin-list specifies all tri-state, open-collector or open-emitter output pins which are disabled by this section. DTG arranges the digital test program so that all ICs that are connected to a bus are tested separately after all non-bused IC tests. Prior to the bused IC tests, DTG inserts a test that checks to see if all DISABLes have successfully freed the bus. If the bus is not free, no bused IC tests for ICs on that bus are performed.

Format

```
.DISAB(output-pin-list);
    ¦
  Test language statements
    ¦
.HOLD(input-pin-list);
.END DISAB;
```

where:	DISAB	Group keyword
	output-pin-list	List of output pins
	HOLD	Command keyword
	input-pin-list	List of input pins
	pin-list	Refer to General Definitions
	END DISAB	Group keywords

Example

```
.DISAB(DATA,ADDR,MREQ,RD,WR,IORQ);
    IL(CLK,BUSRQ);
    IH(CLK);
    IL(CLK);
    IH(CLK);
    IL(CLK);
    .HOLD(BUSRQ);
.END DISAB;
```

Meaning: The IC outputs DATA, ADDR, MREQ, RD, WR, and IORQ are disabled by setting the BUSRQ input low and stepping the CLK input through 2 high states and then holding BUSRQ.

Comments

1. Group-names must be defined prior to being used within the DISABle section.
2. DTG requires that a .HOLD be present in a DISABLe section.
3. Partial disables are used when DTG is unable to disable all pins specified by the DISABle section. A warning is generated so the user may edit the procedure and provide a disable for those outputs which DTG did not disable.

4. DISABle sections containing FAST -- END FAST blocks (SpeedPlus option only) must be written in the format

.DISABle <output-pin-list> USING <fast-pin-list>

Refer to USING.

----------DISJoint----------

General

There are many cases where an IC is connected in a circuit so that an output may be tied to an input. Normally, DTG will disallow a test where this condition exists unless there is a special section included in the library procedure that specifically tests such a circuit condition. In order to tell DTG to allow the output tied to input condition in the MAIN section, a DISJoint statement that defines the condition can be added to the HEADer section.

Format

.DISJ(disj-list);

where: DISJ Command keyword

disj-list Pin-desig_pin-desig_...,pin-desig_...

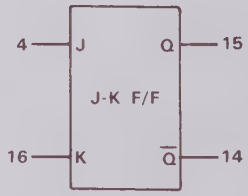
pin-desig Refer to General Definitions

The disj-list must consist of 2 groups of pin-desig's. No pin in either group is allowed to be connected to another pin in the same group but any connetions between groups are allowed.

Example

.DISJ(4 15,14 16);

Meaning: Output pin 15 may be connected to input pin 16 and output 14 to input 4 and a test could be generated. However, if output pin 14 were connected to pin 16 or output 15 to input 4, DTG would not generate a test.



---------DL (default low)----------

General

The discussion of this command directly parallels the discussion of the command DH. The purpose is similar and the format is identical except that DL is used instead of DH to specify a default low for unconnected pins.

----------DU (default unknown)----------

General

For the user who wishes to identify all pins of an IC with a default specification. If neither DH nor DL applies, DU can be used. Any unconnected pins with a DU specifications will not be tested. See the discussion of DH for purpose of default commands. The format of the DU command is identical to that of DH except that DU is used instead of DH to specify non-default pins.

----------ELSE----------

General

Indicates the start of the last case in a case construct when the last case is to be unconditionally included if all preceding cases are false. No conditional is evaluated.

Format

.ELSE;

where: ELSE Case construct keyword

----------ELSE IF----------

General

Indicates the start of each case in a multi-case construct except the first case and the last case if the last case is to be unconditional. The conditionals in each case are evaluated and, if true, the following statements are used. Otherwise, DTG goes to the next case or to the end of the construct.

Format

.ELSE IF conditional;

where: ELSE IF Case construct keyword

conditional Refer to Case Construct definition

----------END----------

General

Indicates the end of a group of instructions.

Format

.END group-ID;

where: END Group Keyword

group-ID CLUST,DISAB,HEAD,HFORCE,LFORCE,INH,INSERT,MAIN,SPEC, or SUB

----------FInish----------

General

Indicates the end of the case construct statements.

Format

```
.FI;
```

where: FI Case construct keyword

Example

```
.IF CONN(A,B);
  ¦
 statements
  ¦
.FI;
```

----------FREE----------

General

This is a function which causes DTG to evaluate the circuit connections of the pins in the pin-expression and to compare the results to the FREE condition. The FREE condition is one in which the pin must not be connected to VCC or GND and in which the pin must have a digital, hybrid or fast nail attached to the node. If all pins in the pin-list satisfy the FREE condition, a true indication is returned to the statement containing the function. A true indication is a value of 1 for the function and a false is a value of 0.

Format

```
FREE(pin-list)
```

where: FREE Function keyword to indicate that a pin-list follows that is to be checked for the FREE condition

pin-list Previously defined in General Definitions

Example

```
.SPEC FREE(2,3);
```

Meaning: If pins 2 and 3 are not connected to VCC or GND and each has a digital, hybrid or fast nail connected to its nodes, then use this SPECial section to test the IC.

----------FREQ----------

General

For systems with the SpeedPlus option, DTG will write a SET CLOCK statement into the .DTP file ahead of every burst that contains one or more FAST sections. .FREQ is included in the header section to specify a range of allowed frequencies for the default clock.

Format

.FREQ low-freq TO high-freq;

where: .FREQ function keyword to indicate that DTG should use default clock parameters. A library procedure containing a .FREQ statement may not contain any .CLOCK statements. Refer to HEADer.

low-freq a value in the range 80 kHz to 2.5 Mhz

high-freq a value in the range 80 kHz to 2.5 Mhz

Example

.FREQ 1MEG TO 2MEG;

Meaning: A SET CLOCK statement using the default timing parameters will be inserted ahead of the burst specifying the lowest possible frequency in the range 1 MHz to 2 MHz.

----------GND----------

General

This is a function which causes DTG to evaluate the circuit connections of the pins in the pin-list to see if they are connected to GND. If all the pins in the list are connected to GND, a true indication is returned to the statement containing the function. True equals the value 1 and false equals the value 0.

Format

GND(pin-list)

where: GND Function keyword to indicate that a pin-list follows that is to be checked for GND.

pin-list Previously defined in General Definitions

Example

.IF GND(G);

Meaning: If the equivalent-pin-list, named G, is connected to GND, then execute the next statement.

----------HEADer----------

General

The HEADer section must always be the first group of statements to appear in a library test procedure. Its purpose is to define the IC and to indicate restrictions and exceptions that apply to its connection in a circuit. The SIZE and INPUT statements are required and at least one of the three output statements (OUTPUT, TRI, OPENC, OPENE) is required. The remaining HEADer statements are used as applicable. The terms .FTEST USING, .FREQ,

.PERIOD, and .TRIGGER are recognized only on systems with the SpeedPlus option.

Format

```
.HEAD;
    .SIZE number;                   !required
    .INPUT(pin-list);               !required
    .OUTPUT(pin-list);              ! } at least one required
    .TRI(pin-list);                 ! } .
    .OPENC(pin-list);               ! } .
    .OPENE(pin-list);               ! } .
    .UNDRIV(pin-list);              !optional
    .DISJ(disj-list);               !optional
    .DH(pin-list);                  !optional
    .DL(pin-list);                  !optional
    .DU(pin-list);                  !optional
    .FTEST USING (F=<fpins>,  -
                  C=<cpins>,  ! !required for SpeedPlus test
                  T=<tpins>); -
    .TRIGGER HI ! LO                !optional for SpedPlus test
    .FREQ (lowest-freq) TO (highest-freq);
                                    !optional for SpeedPlus if no .PERIOD
                                    !or .CLOCK statements
    .PERIOD (lowest-pd) TO (highest-pd);
                                    !optional for SpeedPlus if no .FREQ
                                    !or .CLOCK statements
.END HEAD;
```

where:

HEAD	Group keyword to denote start of header section.
SIZE	Keyword to indicate that IC pin count follows.
number	Integer from 2 through 480 that denotes the IC pin count.
INPUT	Keyword to indicate that a list of input pins follows.
pin-list	Refer to General Definitions.
OUTPUT	Keyword to indicate that a list of output pins follows.
TRI	Keyword to indicate that a list of tri-state pins follows.
OPENC	Keyword to indicate that a list of open collector pins follows.
OPENE	Keyword to indicate that a list of open emitter pins follows. Selects logic level B to use ECL logic levels at these pins.
UNDRIV	Keyword to indicate that a list of undrivable pins follows.
DISJ	Keyword to indicate that a list of input and output pins that may be connected together follows.
disj-list	Pin-desig_pin-desig_...,pin-desig_...

DH — Keyword to indicate that a list of pins follows which will be specified, by default, to be in the high state.

DL — Keyword to indicate that a list of pins follows which will be specified, be default, to be in the low state.

DU — Keyword to indicate that a list of pins follows which will be specified, by default, to be in the unknown state.

FTEST USING — Keywords indicating that the procedure contains sections meant to run on the SpeedPlus subsytem.

<fpins> contains a list of the component pins, separated by spaces, requiring SpeedPlus driver/senor nails.

<cpins> lists pins requiring clock nails.

<tpins> lists pins requiring trigger nails. This list contains all pins mentioned subsequently in the procedure by other USING lists.

TRIGGER HI
TRIGGER LO — Keyword to specify special DTSL default timings. If TRIGGER HI or LO is specified, DTG explicitly defines timing with a single trigger event on the second half of the clock cycle. TRIGGER HI produces a clock going high on the second (triggered) half-cycle, defined by NINC=4 and the parameters

```
SIGNAL(L0, H2)
DST(ED0, S1, D2, S3, T3).
```

If TRIGGER LO is specified, the default timing is the same, but with the clock going low on the first half of the cycle, as defined by

```
SIGNAL(H0, L2)
DST(ED0, S1, D2, S3, T3).
```

FREQ — Keyword to specify a range of allowed test frequencies (test step rates) for SpeedPlus testing. DTG uses this and other timing parameters to write a SET CLOCK statement into the .DTP file. The default D/S rate is twice FREQ. May not appear if PERIOD appears or if a .CLOCK statement is used elsewhere in the procedure.

lowest-freq 20 KHz or higher (but less than highest-freq)
highest-freq 2.5 MHZ or lower (but higher than lowest-freq)

PERIOD — Keyword to specify a range of allowed test step periods for SpeedPlus testing. Equivalent to 1/FREQ. May not appear if FREQ appears or if a .CLOCK statement is used elsewhere in the procedure.

END HEAD — Group keywords to denote the end of the Header section.

Example

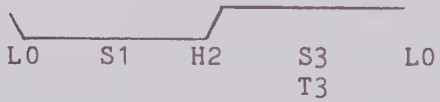
```
/* 74114 DUAL J-K FLIP FLOP */
.HEAD;
   .SIZE 14;
   .INPUT(13 13=CK,2 12=K,3 11=J,4 10=PR,1 1=CLR);
   .OUTPUT(5 9=Q,6 8=NOTQ);
.END HEAD;
```

Comments

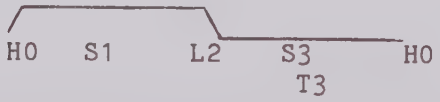
1. DTG supplies timing parameters for all tests, using using either default values or values supplied in the procedure. For standard tests the default values are ITT=4US and IST=1US. A test procedure meant to run on the SpeedPlus hardware requires SpeedPlus timing parameters. If only .FREQ is specified in the library procedure, DTG will write a minimal SET CLOCK statement into the .DTP file containing only the defined frequency (or period) and the terms NINC=4 and SIGNAL(H0, L2). Sense and trigger times default to the hardware-determined default values. Thus, the minimal SpeedPlus default timing is (no .TRIGGER term)

with enable drive (ED) strobes coinciding with the clock transitions, sense strobe (S) 50 to 200 ns and trigger sample strobes (T) 60 to 200 ns ahead of each ED. (Refer to the Test Programming manual for more information on the SET CLOCK statement and its defaults.)

2. If .TRIGGER HI or LO is specified in the HEADer (in addition to .FREQ), DTG writes a SET CLOCK statement containing a DST term in addition to the the NINC and SIGNAL terms. The DST list includes only one trigger (T) term. TRIGGER HI produces the following timing



TRIGGER LO produces the following timing



3. If 2 or more devices must be driven by the SpeedPlus clock simultaneously (e.g., during an INHIBit), DTG can write a default clock to satisfy the requirements of all devices if their library procedures use the DTSL default clock (specified by .FREQ) and their clock requirements do not conflict. DTG attempts to resolve the frequency requirements of 2 or more devices needing the same clock by writing a default clock at the lowest frequency permitted by all of the .FREQ terms. If a procedure specifies TRIGGER HI or LOW, DTG writes a clock with a DST term as in comment 2, unless some other device requires the opposite trigger phase. In the case of conflicting clock requirments, DTG writes no clock statement.

----------HFORCE----------

General

Testing output pins on a bus presents special problems, as any of several ICs connected to the bus could be responsible for a faulty state on the bus. HFORCE is used in the case of a bused node being held in a faulty high state. The pin-list associated with HFORCE identifies the output pins connected to the bus. The bused pins of each connected IC are then forced to assume a high state and the current passing through the bus is measured. This current is compared to the tri-stated current measured before forcing the IC's output high, and the results of this comparison will either indict or exonerate the IC.

Format

```
.HFORCE(output-pin-list);
        |
        |
   test language statements;
        |
        |
.END HFORCE;
```

where: HFORCE Group keyword

output-pin-list List of output pins

pin-list Refer to General Definitions

END HFORCE Group keywords

Example

```
.HFORCE (Q1,Q2)
   IL(D1,D2)
   IH(CLR)
.END HFORCE;
```

Meaning: Pins Q1 and Q2 are output pins connected to a bus which are forced high by setting input pins D1 and D2 low and input pin CLR high.

Comments

1. HFORCE sections are similar to DISABle and INHibit sections with one exception: HOLD commands are neither needed nor allowed within an HFORCE section.

2. HFORCE sections may be written to run on the SpeedPlus subsystem by including a USING <pin list> term after .HFORCE. Refer to USING.

----------HOLD----------

General

When IC input pins are taken high or low by the test program, it is often necessary to maintain that pin-state for the duration of a test sequence (for example, chip enable pins or tristate control pins). The .HOLD command performs this function, and is often used within .DISABle and .INHibit sections to hold a pin's state for use outside (following) the section.

Format

```
.HOLD(input-pin-list);
```

where:	HOLD	Command keyword
	input-pin-list	List of input pins
	pin-list	Refer to General Definitions

Example

```
.DISAB(DATA);
    IH(CE);  /* CE is LOW when ENABLED */
    .HOLD(CE);
.END DISAB;
```

Meaning: The IC is disabled by setting input pins CE high and is held disabled by holding CE high for the duration of the test.

Comments

1. HOLD is used primarily in DISABle and INHibit sections to cause an input pin to remain in the last state to which it was set. HOLD is used for input pins; therefore, it is not needed or allowed in HFORCE or LFORCE sections.

----------IF----------

General

Indicates the start of a case construct. The test language statements that make up the first case will follow. The conditional is evaluated and, if it is true, the first case is evaluated for inclusion in the test program. If it is false, the next case, if one exists, is evaluated.

Format

```
.IF conditional;
```

where:	IF	Case construct keyword
	conditional	Refer to above Case Construct definition

----------IG (input group)----------

Input pin groups may be treated in the same manner as output pin groups. See the description of the keyword OG.

----------INDEPendent----------

General

This is a function which causes DTG to evaluate the circuit condition of the pins in the pin-list to determine if they are connected together. If none of the pins in the pin-list are connected together, then a true indication is returned to the statement containing the function.

Format

```
INDEP(pin-list)
```

where: INDEP — Function keyword

pin-list — Previously defined in General Definitions

Example

```
.IF INDEP(1, 2, 3);
```

Meaning: If there are no connections between any of these pins (1, 2 and 3), execute the next statement.

----------INHibit----------

General

Indicates the start of a section that describes how to inhibit various input pins which may allow glitches to propagate to device pins that are the subject of a test. Feedback loops and free-running clock signal paths are the prime causes of these glitches. The INHibit section consists of a list of pins to be inhibited and a group of statements that define how the state of the various input pins is arrived at to effect the inhibit condition. A HOLD statement would be included to indicate the input pins that must be held in the inhibit state for the duration of the test.

Format

```
.INH(output-pin-list);
    |
    |
  test language statements
    |
    |
  .HOLD(input-pin-list);
.END INH;
```

Comments

1. The formats of the INHibit section and the DISAB section are similar except for the keyword itself. Therefore, the explanations presented in the DISAB paragraph apply here. The INHibit section should follow the DISABle section when both are used.

2. DTG requires a .HOLD statement in an INHibit section.

3. INHibit sections may be written to run on the SpeedPlus subsystem by including a USING <pin list> term after .INH. Refer to USING.

----------INSERT----------

General

A program segment may be written as an INSERT section and included in the library procedure prior to the MAIN and SPECial sections. This INSERT could be selectively or absolutely inserted into the test procedure depending on how it is called. (Refer to "Call Insert" and "CHOOSE Insert.) The INSERT may be written to accept arguments from an argument list provided with the call. It may also be written to be inserted with no argument list.

Format

1. No arguments

```
.INSERT name;
      ¦
      ¦
   insert string
      ¦
      ¦
.END INSERT;
```

2. With arguments

```
.INSERT name=n;
   xxxx?dxxxx;
   xxxx?dxxxx;
      ¦
      ¦
.END INSERT;
```

where:

	INSERT	Group keyword
	name	Character string to identify a particular INSERT
	n	Quantity of arguments in string; digit from 1 through 9
	?	Key character to indicate an argument call follows
	d	Digit from 1 through 9 to identify which argument to use in this text.

END INSERT Group keywords

Example

```
.INSERT FOUR;/*FINISH OP CODE CYCLE */
        IG(CL=CL+D'1') OL(MREQ,RD);
        IG(CL=CL+D'1') OH(MREQ,RD,M1) OI(ADDR) OL(RFSH);
        IG(CL=CL+D'1') OL(MREQ);
.END INSERT;
```

Meaning: The three statements in the INSERT group may be inserted into a test procedure as a result of a call INSERT or CHOOSE I in the library procedure.

----------KNOWN----------

General

There may be conditions in a procedure where it is not possible to know, at some point, whether the state of a pin can be determined by the DTG task. Certain conditions must exist before DTG can determine the state of a pin. The pin must be defined in the circuit description and one of the following sets of conditions must be true:

1. An input or output pin must have a nail specified in the circuit description.

2. An input pin (one currently connected via an IC (pin spec) statement) must be in a circuit configuration which allows it to be driven.

3. A pin that is not an input or output must be connected to VCC or GND.

When DTG encounters the KNOWN function, it will check for the above conditions and return a true or a false indication (1 or 0) as the argument of the statement in which it occurred.

Format

KNOWN(pin-list)

where: KNOWN Function keyword

pin-list Previously defined in General Definitions

Example

```
.IF KNOWN(9,10);
    .CHOOSE I(TEST1, TEST2);
.FI;
```

Meaning: If the state of both pins 9 and 10 of the IC can be determined by DTG, then choose one of the macros and insert it into the program. Otherwise, skip the insertion of a macro.

----------LET----------

General

The calculation of pin lists is usually done by DTG so that, when a pin that is not defined in the circuit description is included in an expression, an error occurs and the calculation is not completed. If it is desired to make a calculation without regard to circuit conditions, a bit string can be defined by a DECLARE BSTRING statement and then used in a LET statement. DTG will not check the circuit description for pins in expressions that occur in a LET statement. A LET statement is allowed in any group except the HEADer.

Format

```
LET bit-string-name = expression;
```

where: LET — Command keyword

bit-string-name — Character string that identifies the bit string variable. Long names are truncated to 8 characters by the library.

expression — Refer to General Definitions

Example

```
LET BSA = BSB .OR.  BSA;
```

Meaning: The values of the bits in the bit string BSA are changed to the values resulting from the ORing of BSA with bit-string BSB.

----------LFORCE----------

General

Testing output pins on a bus presents special problems, as any of several ICs connected to the bus could be responsible for a faulty state on the bus. LFORCE is used in the case of a bused node being held in a faulty low state. The pin-list associated with LFORCE identifies the output pins connected to the bus. The bused pins of each connected IC are then forced to assume a low state and the current passing through the bus is measured. This current is compared to the tri-stated current measured before forcing the IC's output low, and the results of this comparison will either indict or exonerate the IC.

Format

```
.LFORCE(output-pin-list);
        |
        |
   test language statements;
        |
        |
.END LFORCE;
```

where: LFORCE Group keyword

output-pin-list List of output pins

pin-list Refer to General Definitions

END LFORCE Group keywords

Examples

```
.LFORCE (DATA, VALID);
    IL(CLK);
    IH(INPUT);
    IH(CLK);
.END LFORCE;
```

Comments

1. LFORCE sections are similar to DISABle and INHibit sections with one exception: .HOLD commands are neither needed nor allowed.

2. For the SpeedPlus subsystem, USING pin-list must be included.

----------MAIN----------

General

The MAIN section of the test library procedure usually becomes the test burst in the test program as a result of the DTG task (the burst may otherwise result from a SPECial section). All the other sections provide definitions and inserts that are used by DTG and the MAIN section in generating the burst. The BURST statement itself is normally not included in the MAIN section because it is automatically inserted by the DTG task. However, if the BURST statement is included in the MAIN section, the insertion of a BURST statement by DTG is deleted. Putting a BURST statement in the MAIN section provides functions other than the CLEAR function. The DTG BURST contains a CLEAR function but a multiple BURST requires an ACTIVE function in the first BURST statement. Also, the intermediate BURST and END BURST statements must be in the MAIN section because DTG has no way of knowing where to insert them. Other functions that may be defined in the BURST statement are those that define BURST timing (ITT, IST and MAXTIME). The final END BURST may or may not be included in the MAIN section. All BURST statements, whether inserted by DTG or by the library routine, have a label inserted by DTG. This label consists of the IC name. If there is more than one BURST, the label consists of the IC name and BURST number (e.g., U5_B2).

If a USING term appears after .MAIN, DTG will establish clock parameters for the burst, based on the clock requirements of all disables, inhibits, forces, and UUT requirement. Preceding the BURST statement in the .DTP file, it will insert a SET CLOCK statement to program the SpeedPlus clock module before the burst begins. It will allocate nails from the SpeedPlus system for use by the declared component pins. Within the burst, it will permit FAST - END FAST blocks in all disables, inhibits, and forces that require SpeedPlus facilities, if SpeedPlus nails are available for them and if their clock requirements are compatible with the clock requirements of the UUT. DTG will expand a .MAIN USING

section if its SpeedPlus requirements can be met; otherwise it will proceed to a standard .MAIN section, if one is present. If the standard .MAIN section is placed first, a subsequent .MAIN USING will be ignored.

Formats

1.
```
.MAIN;
    ¦
    ¦
  test language statements
    ¦
    ¦
.END MAIN;
```

where: MAIN Group keyword

END MAIN Group keywords

2. A test for an IC whose name is U5 and which requires two BURSTs would have the following format in the library:

```
.MAIN;
BURST ACTIVE;
    ¦
  DTSL and Test Language statements
    ¦
END BURST;
BURST;
    ¦
  DTSL and Test Language statements
    ¦
END BURST;
.END MAIN;
```

where: MAIN Group keyword

BURST ACTIVE
END BURST Burst keywords
BURST
END BURST

END MAIN Group keywords

The resulting test procedure is produced by DTG:

```
U5:    BURST ACTIVE;
           ¦
        test language statements
           ¦
       END BURST[BRANCH U5_Z;]

U5_B2: BURST;
           ¦
        test language statements
           ¦
       END BURST;
U5_Z:  CLEAR LGC;
```

When multiple bursts are necessary, the library procedure must contain the first as well as all intermediate BURST statements. The BURST ACTIVE is required because DTG would otherwise insert BURST (CLEAR). The intermediate BURST and END BURST are required because DTG would not otherwise know where the bursts are divided. DTG adds the labels to all bursts. If a failure occurs at run time in the first burst, the second burst will not be performed. If a failure occurs in the second burst there will be two statistics which the user must recognize as being for the same device.

Example

```
.MAIN;
    IC(A,B) OS(0) IH(A) IL(B) OH(0);
    IH(B) OL(0);
    IL(A) OH(0);
    IL(B);
    ID(A,B) OI(0);
.END MAIN;
```

Meaning: The 5 statements in the group will be inserted into the test procedure and will become the burst that tests the IC. If this procedure is being used for IC U4 from a circuit description file, the BURST statement U4: BURST; is inserted at the beginning and the statement END BURST; is inserted at the end.

Comments

1. The only arguments allowed to a BURST statement used in a library routine are:

   CLEAR
   ACTIVE
   IST
   ITT
   MAXTIME
   VTT (only in .MAIN USING or .SPEC USING)

2. .MAIN sections may be written to run on the SpeedPlus subsystem by including a USING <pin-list> term after .MAIN. Refer to USING. Within a .MAIN USING section, the keyword BURST may be used to specify parameters for standard digital testing, as well as the parameter VTT=<trigger-threshold-voltage>. VTT defines the transition voltage from logic low to logic high for triggering purpose. BURST VTT= is local to the BURST. The value of VTT in the next BURST reverts to the value given in an ASSIGN statement (in .DTP file).

3. Standard pins automatically remain in the BURST ACTIVE state during FAST sections of the burst. That is, all the standard pin context remains unchanged from FAST; to END FAST;. However, an IC or OS statement must be repeated after the FAST section to maintain connection of all active pins. Pin states could also be repeated for added assurance.

4. FLOOPs are restricted to 64K iterations.

5. EXITIF must appear no fewer than 5 test steps ahead of END FLOOP.

6. When OPENC or OPENE is specified for outputs in the header, the system automatically applies pull-up (PU) or pull-down (PD) resistors in the test. There is an exception to this which must be considered with FAST sections. If the BURST begins with a FAST section, or if a hybrid nail is used in the FAST section, the system does not automatically apply PU/PD. PU/PD statements must be explicitly written into the procedure.

----------NP (no preference)----------

General

Indicates no preference. DTG is not to be concerned, when evaluating the next statement, with the state of any pin presently defined. The state which is naturally imposed by the circuit and test configuration is included.

Format

NP(pin-list)

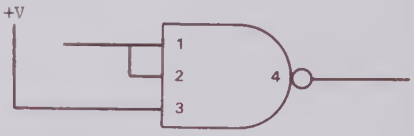
where: NP — Function keyword

pin-list — Refer to General Definitions

Example

These library (source) statements:

```
IC(INI-IN3) OS(OUT4) IH(IN1-IN3) OL(OUT4);
IH(IN1) IL(IH2);
IL(IN1,IN2) OH(OUT4);
IH(IN1) NP(IN2) OH(OUT4);
```



Produce these test program like these;

```
IC(1-3) OS(4) IH(1-3) OL(4);
                  /* THIS LINE REJECTED */
IL(1,2) OH(4);
IH(1,2);
```

DTG accepts the first line from the library as it is. It rejects the second line because it cannot drive inputs 1 and 2 to different states (they are tied together). The third line is accepted, however, because DTG is allowed by NP(IN2) to assign a possible state to input 2, namely, the same state as pin 1. DTG does not repeat the term OH(4) in the last line because this term is already implicit in the nail context.

----------OG (output group)----------

General

A group of output pins may have been given a group name by a DECLARE GROUP statement. This group of pins can be assigned states by the library programming element called output group (OG). When the group name was given, the pins were listed in such an order that they could be referenced by an address consisting of up to 32 bits. The order of listing the pins is with the least significant address bit corresponding to the rightmost pin listed and the

most significant address bit corresponding to the leftmost pin listed. This is an important feature because of the fact that many ICs interface with a bus and are controlled by the bus addresses. Pin states are specified as follows:

Formats

```
      OG(group-name=expression)
   or
      OG(group-name=expression, group name=expression...)
   or
      OG(group-name=label:group-name)
```

where: OG — Function keyword

group-name — Name assigned in DECLARE GROUP statement

expression — Refer to General Definitions

label — A label used on a previous statement

Example

Assume a DECLARE GROUP statement has defined a group of 6 pins with the group name DAT2 and the pin list (3, 1, 5-8). Then the following statement defines the pin states of the pins.

```
OG(DAT2=B'011000');
```

The resulting pin states are:

Pin	3	1	5	6	7	8
state	Lo	Hi	Hi	Lo	Lo	Lo

The statements that will be inserted in the DTG-developed procedure in place of this statement are as follows:

```
OL(3,6,7,8)
OH(1,5);
```

The many forms of the OG statement are illustrated by the following sequence of commands which modify the pin states of the preceding command, using the above state as the leading context.

	Resulting Pin States						
	pins in group						object
Sequential statements	3	1	5	6	7	8	group
Starting context of DAT2	0	1	1	0	0	0	DAT2
OG(DAT2=D'3');	0	0	0	0	1	1	DAT2
OG(DAT2=DAT2 + 0'4');	0	0	0	1	1	1	DAT2
OG(DAT2=DAT2.OR. X'F');	0	0	1	1	1	1	DAT2
OG(DAT3=DAT2);	0	0	1	1	1	1	DAT3

----------OU (output unknown)----------

General

With some ICs, such as ROMs, it is difficult to anticipate an output state for a given input. The unknown pin states are specified by the OU statement. At debug time, the learn feature is used to obtain the correct output states. These found states are merged into the test program.

Format

OU (pin-list)

where:	OU	Function keyword
	pin-list	Previously defined in General Definitions

Example

OU(D1,D2,D3,D4)

Meaning: The pin states of the pins named D1,D2, D3, and D4 are unknown when writing the library procedure. These pin states will be learned at debug time and will be replaced in the program with OL and OH statements.

----------OV (output value)----------

General

Indicates the output value of the indicated pins is to be determined by analyzing the expressions for each pin. These expressions are relational with respect to inputs that have been defined in the current, or preceding, statements. This technique results primarily from the use of the NP, PH, and PL statements where the actual input pin states are not known until DTG time.

Formats

OV(output-pin-designation=pin-designation)

or

OV(output-pin-designation=statement-label:pin-designation)

or

OV(output-pin designation=pin-designation.logical operator.pin-designation)

or

OV(multiple combinations of expressions in the preceding formats, separated by commas)

where:	OV	Function keyword
	output-pin-designation	Name of output pin to be assigned a value
	pin-designation	Name of pin whose value is assigned
	statement-label	Label of statement with pin-designation
	logical operator	OR, AND or NOR

Examples

1. OV(OUT=IN)

   Meaning: The output takes the state that the input presently has.

2. OV(OUT=L5:IN)

   Meaning: The output takes the state the input had just prior to the statement labeled L5.

3. OV(OUT1=L7:IN1.OR.L8:IN2,OUT2=L9:IN3.AND.L10:IN4)

   Meaning: Output OUT1 takes the state resulting from the "logical OR" of inputs IN1 just prior to the statement labeled L7 and IN2 just prior to the statement labeled L8. Output OUT2 takes the state resulting from the "logical AND" of inputs IN3 just prior to the statement labeled L9 and IN4 just prior to the statement labeled L10.

----------PHn (preferred high)----------

General

Indicates a preference to have all indicated pins in the high state. A minimum of n pins must be drivable to the high state for the statement to be acceptable for inclusion in the test program. When used in INHibits, DISABles or FORCEs, PH1 must be specified. No other n is allowed.

Format

PHn(pin-list)

where: PH Function keyword

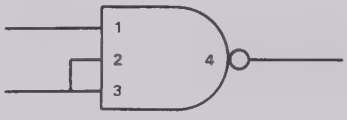
n Integer

pin-list Previously defined in General Definitions

Example

These source statements:

```
IL(IN3) PH(IN2);
IL(IN3) PH1(IN2);
IL(IN3) PH1(IN2,IN1);
IH(IN3) PH(IN2);
```



Produce test program statements like these;

```
IL(2);
    /*THIS LINE REJECTED*/
IH(1);
IH(2);
```

Meaning: Because inputs 2 and 3 are tied together, DTG evaluates the first line simply with IL(2). The 'preferred' PH(IN2) term is rejected in favor of the 'required' IL(3) term, but DTG uses the lowest nail number when 2 or more

IC pins are tied -- IL(2) actually comes from IL(3). The second line requires that inputs 2 and 3 be driven to different states, so the step is rejected. In the third line, DTG produces IH(1) from PH1(2,1); input 2 is already being driven low along with 3. Finally, input 3 is required and input 2 is preferred high in line 4. As before, the lower number of the 2 tied pins is specified. If the node name at pin 3 has VCC as its first 3 characters, the system automatically considers pin 3 high.

----------PLn (preferred low)----------

General

Indicates a preference to have all indicated pins in the low state. A minimum of n pins must be drivable to the low state for the statement to be acceptable for inclusion in the test program. When used in inhibits, disables, or forces, PL1 must be specified. No other n is allowed.

Format

PLn(pin-list)

where: PL — Function keyword

n — Integer

pin-list — Previously defined in General Definitions

--------QUIT--------

General

QUIT is used to terminate the evaluation and selection process for the present IC and to insert a message into the report file indicating the IC is not tested. A quoted string, if included in the command, is treated as a SAY message.

Format

.QUIT "message";

where: QUIT — Command keyword

message — Any string enclosed within quotation marks

Example

```
.INSERT SHOW=1;
    .IF UNDEF (?1);
    .QUIT "Not executed.  Missing argument.";
        |
      statements for SHOW insert;
        |
.END INSERT;
```

Meaning: If the argument (?1) to the insert routine is not defined at DTG time, processing is terminated and the QUIT message is displayed.

--------REPEAT--------

General

This command set makes it possible to learn conveniently the response of a device to a repetitive input sequence. It expands a designated sequence of statements into a series of sequences for execution.

Format

```
.REPEAT ns 'count';
    ¦
    ¦
  DTSL and Test Language statements
    ¦
    ¦
.END REPEAT;
```

where: REPEAT — Command keyword.

ns — A letter that indicates the number system used in count; it may be one of the following:

- D for decimal
- B for binary
- O for octal
- X for hexidecimal

count — The number of times the REPEAT sequence is to be expanded, expressed in the number system specified for ns.

END REPEAT — Command keywords.

Example

```
.REPEAT D'3';
    IH(1) OL(2);
    IL(1);
.END REPEAT;
```

Meaning: The test sequence will be expanded and written into the test program as follows:

```
IH(1) OL(2);
IL(1);
IH(1);
IL(1);
IH(1);
IL(1);
```

----------SAY----------

General

The SAY statement is used in MAIN or SPEC sections and causes DTG to insert the message into the digital report file (.MSG). SAY may not be used within INHibit, DISABle, LFORCE, or HFORCE sections.

Format

```
.SAY "message";
```

where: SAY — Command keyword

message — Any character string excluding quotes (")

----------SETINDependent----------

General

This is a function which causes DTG to evaluate the circuit connections of the IC pins in the set-list to see if there is any pin from one group that is connected to any pin in another group. If no such pins are connected together, a true indication is returned.

Format

```
SETIND(set-list)
```

where: SETIND — Function keyword

set-list — List of pins in group A, list of pins in group B,....

Example

```
.IF SETIND(1 2 3,4 5 6);
```

Meaning: If pin 1, 2 or 3 is connected to pin 4, 5 or 6, then jump to the next .ELSE IF, .ELSE, or .FI statement.

----------SPECial----------

General

Indicates the start of a special conditions section. A SPECial section is simply an alternative to the MAIN section. Because there are some ICs which are often used in standard wiring configurations, the test procedure for an IC could be more efficient if it were written specifically for this configuration. The SPECial instruction allows for the providing of a separate routine in the test library which can be used when the "standard" wiring configuration is found by the conditional expression in the instruction, i.e., if the circuit description contains the connection defined in the conditional.

Format

```
.SPEC conditional;
     ¦
     ¦
  Test language statements
     ¦
     ¦
  [.SAY "message";]
.END SPEC;
```

where: SPEC — Keyword that denotes the start of a test procedure section designed to write a test for an IC connected in the "standard" configuration

conditional — System function or any equation whose expression can be evaluated by the system

SAY — Keyword that denotes that a message follows

message — Any character string that does not include quotes (")

END SPEC — Group keywords

Example

```
.SPEC VCC(B).AND.VCC(C);
      .SAY"WITH BOTH THE B AND C INPUTS TIED HIGH THE OUTPUTS
SHOULD ALWAYS BE LOW";
      IC(A1,A2,A3,A4) OS(Y1,Y2,Y3,Y4)
      PH(A1,A2,A3,A4) OL(Y1,Y2,Y3,Y4);
      PL(A1,A2,A3,A4) OH(Y1,Y2,Y3,Y4);
.END SPEC;
```

Comments

1. The .SAY "message" statement is generally included to indicate that the SPEC section was used to generate the test.

2. .SPECial sections may be written to run on the SpeedPlus subsystem by including a USING <pin list> term after .SPEC. Refer to USING. Within a .SPEC USING section, the keyword BURST may be used to specify parameters for standard digital testing, as well as the parameter VTT11=<trigger-threshold-voltage>. VTT defines the transition voltage from logic low to logic high for triggering purpose. BURST VTT= is local to the BURST. The value of VTT in the next BURST reverts to the value given in an ASSIGN statement (in the .DTP file).

3. Standard pins automatically remain in the BURST ACTIVE state during FAST sections of the burst. That is, all the standard pin context remains unchanged from FAST; to END FAST;. However, an IC or OS statement must be included after the FAST section to maintain connection to all active pins. Pin states could also be repeated for added assurance.

4. FLOOPs are restricted to 64K iterations.

5. EXITIF must appear no fewer than 5 test steps ahead of END FLOOP.

6. When OPENC or OPENE is specified for outputs in the header, the system automatically applies pull-up (PU) or pull-down (PD) resistors in the test. There is an exception to this which must be considered with FAST sections. If the BURST begins with a FAST section, or if a hybrid nail is used in the FAST section, the system does not automatically apply PU/PD. PU/PD statements must be implicitly written into the procedure.

----------SUBroutine----------

General

Indicates the start of a subroutine block whose identification is "name." The subroutine may be called any place in the test program by a "GOSUB" statement. It may also be called as the result of DTG inserting a subroutine call statement in place of a ".CHOOSE C(choice-list)" statement. The test language statements must include IC and OS statements because the context of pin conditions is zero at the beginning of a subroutine.

Format

```
.SUB name;
    |
    |
  Test language statements
    |
    |
.END SUB;
```

where: SUB — Keyword to denote start of a subroutine

name — Subroutine name

END SUB — Keywords to denote end of subroutine

Comments

1. Subroutines must be located after the HEADer section and before the section in which they are called.

2. The subroutine test language statements must include IC and OS statements because the software assumes zero context at the beginning of a subroutine. The context after the subroutine reverts back to the state of the pins just before the subroutine call.

3. Subroutines cannot be called from nor defined in FAST blocks (SpeedPlus option only).

----------TG (trigger group)----------

General

A group of SpeedPlus trigger pins may have been given a group name by a DECLARE GROUP statement. This group of pins can be assigned states by the library programming element called output group (TG). When the group name was given, the pins were listed in such an order that they could be referenced by a bit string consisting of up to 32 bits. The order of listing the pins is with the least significant address bit corresponding to the rightmost pin listed and the most significant address bit corresponding to the leftmost pin listed. Pin states are specified as follows:

Formats

TG(group-name=expression)

or

TG(group-name=expression, group name=expression...)

or

TG(group-name=label:group-name)

where: TG Function keyword

group-name Name assigned in DECLARE GROUP statement

expression Refer to General Definitions

label A label used on a previous statement

Note that trigger terms are not incremental; that is, a trigger statement in one step does not carry into following steps.

----------UG (unknown group)----------

General

When testing complex ICs such as ROMs, it may be difficult to anticipate the pin states of pin groups. The Learn Mode feature of the Run-time system can be used to define the proper states for the group. When Learn Mode is used, it tests the specified burst a specified number of times and the result is reported with a percent correlation. The result is merged into the test program (.TPG) and the unknown states become specified in IG and OG format.

Format

UG(group-name)

where: UG Function keyword

group-name A name specified in a DECLARE GROUP statement

Example

UG(DATOUT)

Meaning: The pin states of the group of pins declared as DATOUT are unknown. The pin states can be learned and this statement element then replaced with UG(DATOUT = ...).

----------UNDEFined----------

A function is provided to allow for the checking of an element to see if it is UNDEFined. This function is primarily for use in INSERTs to determine if an argument is UNDEFined in the calling statement. The function is legal, however, for any reasonable application. DTG will return a true indication when any of the items in the UNDEFined list are UNDEFined.

Format

```
UNDEF(def-list)
```

where: UNDEF Function keyword

def-list One or more conditions separated by commas

Example

```
.INSERT SMPL=3;
    .IF UNDEF(?1, ?2, ?3);
        .ELSE
          |
    .FI
.END INSERT;
```

Meaning: Abort the performing of the INSERT if any of the three arguments is not defined in the calling statement.

---------USING---------

General

On systems with the SpeedPlus option, USING indicates that DTG should allocate resources from the SpeedPlus subsystem for use in the section and that it should permit the section to contain FAST blocks of test statements. Within the FAST block may appear only those pins' names that have been declared in the USING statement's pin list. The sections INHibit, DISABle, HFORCE, LFORCE, SPECial, and MAIN can be specified for the SpeedPlus hardware with USING. In addition, the FTEST statement in a HEADer section must include a USING statement with slightly different syntax than that described here (refer to HEADer).

Format

```
USING ( clk, F=<fpins>, T=<tpins>);  /* Timing from a named .CLOCK */
```

or

```
USING ( C=<cpins>, F=<fpins>, T=<tpins>); /* Uses implicit clock */
```

where: USING SpeedPlus keyword

clk Name of a .CLOCK statement used to define timing for the section. If clk does not appear, the implicit (default) clock defined by the term .FREQ in the HEADer section is used.

cpins Names of component pins requiring SpeedPlus clock driver or clock sync nails.

fpins Names of component pins requiring SpeedPlus driver/sensors.

tpins Names of component pins requiring SpeedPlus trigger nails.

----------VCC----------

General

This is a function which causes DTG to evaluate the circuit conditions for the pins in the pin-list to see if they are connected to VCC. If all the pins in the list satisfy this condition, a true indication is returned to the statement containing the function.

Format

VCC(pin-list)

where: VCC Function keyword

pin-list Previously defined in General Definitions

Example

```
.SPEC VCC(7) .AND. VCC(14);
```

Meaning: If pin 7 and pin 14 are connected to VCC, then perform this SPECial section to test the IC.

6.4 SAMPLE TEST PROCEDURE

This section presents an example of a library test procedure for the 7454 IC. The library procedure (7454.DTS) is annotated in Figure 6-16 to provide an explanation of each of the statements in the procedure. This procedure is used to generate a test program for a 7454 that is connected in a circuit with 2 of its input gates tied together as shown in Figure 6-17. The resulting digital test program (7454.DTP) is shown in Figure 6-18. A comparison of the library procedure with the test program shows how DTG uses the DTSL statements to structure the test for the particular circuit connections.

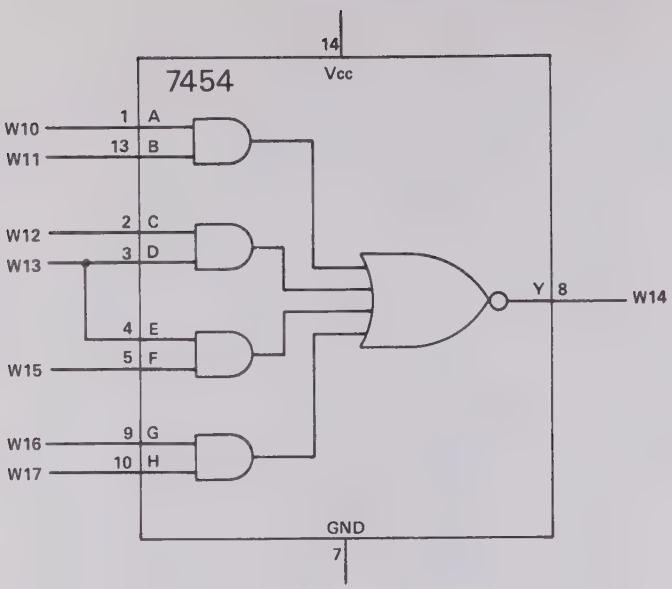
The digital test program (.DTP) must be edited (manually or with the .ATO or .ATX option) for proper instrumentation statements and merged with the .ATP before it is translated. This finished test program (.TPG or .TPX) is illustrated in Figure 6-19.

During DTG, a report file (.MSG) is generated which gives statistics about the test. This report is illustrated in Figure 6-20. An explanation of this report is given in the Test Set Preparation manual.

Digital Test Library program for 7454 IC File name 7454.DTS

```
/*7454*                                  ! IC Type
/*4-WIDE AND OR INVERT*/                 !
                                         !
.HEAD;                                   ! Indicates beginning of header section
                                         ! which defines the IC pin configuration.
                                         !
   .SIZE 14;                             ! IC has 14 pins.
                                         !
   .INPUT (9=IN1A,10=1N1B,13=IN2A,       ! Input pins are 9,10,13,1,2,3,4, and 5.
           1=IN2B,2=IN3A,3=IN3B,4=IN4A,  ! They are named IN1A,IN1B,IN2A,IN2B,IN3A,
           5=IN4B,);                     ! IN3B, IN4A, and IN4B, respectively.
                                         !
   .DH (IN1A,IN1B,IN2A,IN2B,IN3A,        ! Any of the input pins are to be
        IN3B,IN4A,IN4B);                 ! considered high if not connected in the
                                         ! circuit description so that a test may
                                         ! be generated.
                                         !
   .OUTPUT (8=OUT);                      ! Output pin is 8 and is named OUT.
                                         !
.END HEAD;                               ! Indicates end of header section.
                                         !
.INH (OUT);                              ! In the case where the output of this IC
         PL1 (IN1A,IN1B,IN2A,IN2B,IN3A,  ! is to be inhibited, all the inputs are
              IN3B,IN4A,IN4B);           ! preferred to be low and are to be held
        .HOLD(IN1A,IN1B,IN2A,IN2B,IN3A,  ! in the low state for the duration of
              IN3B,IN4A,IN4B);           ! the inhibit.
.END INH;                                !
                                         !
.MAIN;                                   ! Indicates beginning of the main section
                                         ! of the test.
                                         !
IC(IN1A,IN1B,IN2A,IN2B,IN3A,IN3B,IN4A,   ! Instruction to enable all inputs and the
   IN4B) OS(OUT)                         ! output.
                                         !
/*WORKS ON ONE AND GATE AT A TIME*/      ! Descriptive comment
                                         !
   IH(IN1A,IN1B) PL(#IN1A,IN1B) OL(OUT); ! These four steps of the burst will apply
   IH(IN2A,IN2B) PL(IN1A,IN1B);          ! a logic Hi to both inputs of each AND
   IH(IN3A,IN3B) PL(IN2A,IN2B);          ! gate one at a time.  It tells ATG that
   IH(IN4A,IN4B) PL(IN3A,IN3B);          ! it is preferred (true if circuit allows)
                                         ! that all inputs to the other AND gates be
                                         ! Lo.  The output should stay Lo.
                                         !
/*WORKS ON HALF OF EACH AND GATE*/       ! Descriptive comment
                                         !
   PL1(IN1A,IN1B) PL1(IN2A,IN2B)         ! Tells ATG that it is preferred that one
   PL1(IN3A,IN3B) PL1(IN4A,IN4B) OH(OUT);! input to all gates be Lo and the
                                         ! output go Hi.
                                         !
/*TRIES TO PULL ALL INPUTS HIGH TO CHECK ! Descriptive comment
FOR  SA0*/                               !
                                         !
   PH(#) OV(OUT=.NOT.((IN1A.AND.IN1B)    ! Tells ATG that all inputs are preferred
            .OR.(IN2A.AND.IN2B).OR.      ! Hi. The conditional says, providing at
            (IN3A.AND.IN3B).OR.          ! least one pair of inputs is Hi, the
            (IN4A.AND.IN4B)));           ! output will be Lo.
                                         !
/* CHECKS EACH AND GATE SEPARATELY*/      ! Descriptive comment
                                         !
   IL(IN1A) IH(IN1B) PL1(IN2A,IN2B)      ! These 8 steps apply a logic LO and a
   PL1(IN3A,IN3B) PL1(IN4A,IN4B) OH(OUT);! logic Hi to the 2 inputs of one gate and
                                         ! tells ATG that all other inputs are
   IL(IN1B) IH(IN1A);                    ! preferred Lo.  This is repeated for each
                                         ! gate.  The output should remain Hi.
   IL(IN2A) IH(IN2B) PL1(IN1A,IN1B);     !
   IL(IN2B) IH(IN2A);                    !
                                         !
   IL(IN3A) IH(IN3B) PL1(IN2A,IN2B);     !
   IL(IN3B) IH(IN3A);                    !
                                         !
   IL(IN4A) IH(IN4B) PL1(IN3A,IN3B);     !
   IL(IN4B) IH(IN4A);                    !
                                         !
   ID(#) OI(#);                          ! Disconnects all inputs and the output.
                                         !
.END MAIN;                               ! Indicates end of main part of test.
```

Figure 6-16. Sample Digital Test Library Procedure for 7454 IC.




```
/*Circuit description for 7454 Gate Card   File Name is 7454GATE.CKT*/

%CIRCUIT;

/*7454 with Inputs of 2 Gates Tied Together*/

  W10  EXT   ,IC1.1;
  W11  EXT   ,IC1.13;
  W12  EXT   ,IC1.2;
  W13  EXT   ,IC1.3 IC1.4;
  W14  EXT   ,IC1.8;
  W15  EXT   ,IC1.5;
  W16  EXT   ,IC1.9;
  W17  EXT   ,IC1.10;
  VCC  EXT   ,IC1.14;
  GND  EXT   ,IC1.7;
  IC1  7454  W10,W12,W13,W13,W15,,GND,W14,W16,W17,,,W11,VCC;

/*No VALUE Section is Required with a Digital Only Circuit*/

%ADAPTOR:2270;

  D16   =IC1.1;
  D17   =IC1.2;
  D18   =IC1.3 IC1.4;
  D19   =IC1.5;
  D20   =IC1.8;
  D21   =IC1.9;
  D22   =IC1.10;
  D23   =IC1.13;
  D24   =IC1.7;
  D25   =IC1.14;
```

Figure 6-17.
Sample Circuit for 7454 IC.

```
/* DTG3V83 SC:GATE7454.DTP       3-SEP-80 10:54 */

DECLARE BSTRING FAIL(5);
SET PIO(0) HRLY(CLOSE 3);       /* GROUND PCB */
/* TYPICAL POWER SKELETON
SET PIO(1) HRLY(CLOSE 1);       /* ATTACH UPS TO PCB */
SET DCV UPS(4) V=5 I=0.5 DLY=20M
        [WRITE 'UPS(4) DID NOT SETTLE%NL%'; BRANCH DDONE;];
*/

ASSIGN LGC VIHA=4.5 VILA=0.0 VOHA=2.4 VOLA=0.8 LVLA(16 - 25);
IC1: BURST;     /* (7454) */

/** TEST PROGRAM FOR IC IC1 (7454)
PIN NAIL COMMENT
  1 16
  2 17
  3 18
  4 18
  5 19
  6 NC  (ANALOG)
  7 24  (GND)
  8 20
  9 21
 10 22
 11 NC  (ANALOG)
 12 NC  (ANALOG)
 13 23
 14 25  (VCC)
 **/
/*WORKS ON ONE AND GATE AT A TIME*/
        IC(16-19,21-23) IL(16-19,23) IH(21,22) OS(20) OL(20);
        IL(21,22) IH(16,23);
        IL(16,23) IH(17,18);
        IL(17) IH(19);
/*WORKS ON HALF OF EACH AND GATE*/
        IL(18,19) OH(20);
/*TRIES TO PULL ALL INPUTS HIGH TO CHECK FOR  SA0*/
        IH(16-19,21-23) OL(20);
/*CHECKS EACH AND GATE SEPATATELY*/
        IL(16-19,21,23) OH(20),
        IL(22) IH(21);
        IL(21) IH(16);
        IL(16) IH(23);
        IL(23) IH(18);
        IL(18) IH(17);
        IL(17) IH(19);
        IL(19) IH(18);
        ID(16-19,21-23) OI(20);
        END BURST; /* IC1 */
/* Begin Bussed IC Tests */

DDONE:  WRITE 'DIGITAL TESTING FINISHED';
/*
        CLEAR UPS(4); /* TURN OFF POWER */
        DELAY 20M;      /* WAIT FOR CAPACITORS */
        SET PIO(1) HRLY(OPEN 1); /* DETACH UPS(4) */
*/
        SET PIO(0) HRLY(OPEN 3);
        END;
/* ATG3V604 GATE7454     3-SEP-80 10:54         PAGE 1.1 */

/* THIS PROGRAM GENERATED USING CIRCUIT DESCRIPTION GATE7454 */
 DECLARE RVAL;  DECLARE CBETA;
 DECLARE VVAL1; DECLARE VVAL2;
 DECLARE IVAL1; DECLARE IVAL2;
 DECLARE CSTRING DSTRG(20);
 DECLARE BSTRING FAIL(5);

 DECLARE FAILCONT();
```

Figure 6-18.
Sample Digital Test Program
for 7454 IC Circuit.

```
/* ATG3V604 GATE7454     3-SEP-80 10:54         PAGE 20.1 */


       /* SHORTS TEST */
PRINT:  TEST SHORTS AT( 0: 16-25: 0)
         VMAX=200M R=10
         FAIL(3) [WRITE 'FAILED SHORTS TESTS - TESTING HALTED%NL%';

                  BRANCH DONE;];
       /* CONTINUE ON SHORTS */
       IF FAIL(3) THEN [WRITE 'FAILED SHORTS TESTS - ';
                        WRITE 'CONTINUE WITH DIGITAL%NL%';
                        BRANCH ADONE;];
```

```
/* ATG3V604 GATE7454     3-SEP-80 10:54         PAGE 100.1 */

ADONE:  USE NOFSUB;
        CLEAR SCAN;
        CLEAR MUX;
/* DTG3V83 SC:GATE7454.DTP       3-SEP-80 10:54 */

SET PIO(0) HRLY(CLOSE 3);       /* GROUND PCB */
/* TYPICAL POWER SKELETON
SET PIO(1) HRLY(CLOSE 1);       /* ATTACH UPS TO PCB */
SET DCV UPS(4) V=5 I=0.5 DLY=20M
        [WRITE 'UPS(4) DID NOT SETTLE%NL%'; BRANCH DDONE;];
*/

ASSIGN LGC VIHA=4.5 VILA=0.0 VOHA=2.4 VOLA=0.8 LVLA(16 - 25);

IC1: BURST;     /* (7454) */

/** TEST PROGRAM FOR IC IC1 (7454)
PIN NAIL COMMENT
  1 16
  2 17
  3 18
  4 18
  5 19
  6 NC  (ANALOG)
  7 24  (GND)
  8 20
  9 21
 10 22
 11 NC  (ANALOG)
 12 NC  (ANALOG)
 13 23
 14 25  (VCC)
 **/
/*WORKS ON ONE AND GATE AT A TIME*/
        IC(16-19,21-23) IL(16-19,23) IH(21,22) OS(20) OL(20);
        IL(21,22) IH(16,23);
        IL(16,23) IH(17,18);
        IL(17) IH(19);
/*WORKS ON HALF OF EACH AND GATE*/
        IL(18,19) OH(20);
/*TRIES TO PULL ALL INPUTS HIGH TO CHECK FOR  SA0*/
        IH(16-19,21-23) OL(20);
/*CHECKS EACH AND GATE SEPATATELY*/
        IL(16-19,21,23) OH(20);
        IL(22) IH(21);
        IL(21) IH(16);
        IL(16) IH(23);
        IL(23) IH(18);
        IL(18) IH(17);
        IL(17) IH(19);
        IL(19) IH(18);
        ID(16-19,21-23) OI(20);
        END BURST; /* IC1 */

/* Begin Bussed IC Tests */

DDONE:  WRITE 'DIGITAL TESTING FINISHED';
/*
        CLEAR UPS(4); /* TURN OFF POWER */
        DELAY 20M;      /* WAIT FOR CAPACITORS */
        SET PIO(1) HRLY(OPEN 1); /* DETACH UPS(4) */
*/
        SET PIO(0) HRLY(OPEN 3);

/* DIGITAL TESTS INSERTED HERE */
DONE:   /* END OF DIGITAL TESTS */


/* ATG3V604 GATE7454     3-SEP-80 10:54         PAGE 240.1 */

        IF FAIL(1) THEN BRANCH FDONE;
        IF FAIL(2) THEN BRANCH FDONE;
        IF FAIL(3) THEN BRANCH FDONE;
        BRANCH PDONE;
FDONE:  LET DSTRG=DATE();
        WRITE ID=MESFILE
        '%036% %NL%%S% %NL%%035%
'       DSTRG;
        WRITE ID=MESFILE
                '%036% BOARD GATE7454 FAILED%035%
'       ;
PDONE:  END;


/* ATG3V604 GATE7454     3-SEP-80 10:54         PAGE 250.1 */

 SUBROUTINE FAILCONT();
        BITSET(FAIL,1);         /* SET FAIL(1) ON FAILURE */
        WRITE ID=MESFILE '%037%%S%%036% FAILED CONTINUITY%NL%' FCMSG;
        WRITE ID=MESFILE '  BETWEEN %S%%035%%NL%' FDMSG;
        IF FPMSG<>'' THEN [WRITE ID=MESFILE '%S% %NL%' FPMSG;
                           WRITE ID=LOGFILE '=%S%' FPMSG;];
 END SUBROUTINE FAILCONT;
```

Figure 6-19. Sample Complete Test Program for 7454 IC Circuit.

```
ATG Report On Digital Section SC:GATE7454.DTP 3-SEP-80 10:54
(DTG3V83) Libraries:   DIGLIB610.DTL
--------------------------------- ----------------
BOARD STATISTICS SUMMARY

       IC count                  1
       Undrivable ICs            0
       Lowest nail used         16
       Highest nail used        25


       PIN RESULTS FOR IC1 (7454)
       Pin Type Nail Tested? Comments
        1 IN   16      YES
        2 IN   17      YES
        3 IN   18      YES
        4 IN   18      YES
        5 IN   19      YES
        6 ANA
        7 GND  24
        8 OUT  20      YES
        9 IN   21      YES
       10 IN   22      YES
       11 ANA
       12 ANA
       13 IN   23      YES
       14 VCC  25
  14 Test steps generated for IC1(7454).


**     1 ICs Tested
**     0 ICs Untested

  0 Bus Tests were written.
```

Figure 6-20. Sample Message File for 7454 IC Circuit.

Analog Component Library User's Guide - Section 7

7.1 GENERAL

The Circuit Description Compiler recognizes a number of primitive analog components, such as resistors, inductors, and transistors, as shown in Figure 2-2. It can also recognize other, complex analog components, if entries for them appear in the Analog Component Library. A complex analog component is one that consists of some number of primitive devices. Typically it contains a number of devices connected together to form a functional unit, such as a filter or an amplifier, as illustrated in Figure 2-2.

Library descriptions of such components serve 3 purposes. First, they provide information about the devices contained in a complex component, so ATG's circuit analysis can take account of them in calculating impedances at nodes connected to the device. This aids ATG in writing correct tests for other devices connected to the complex component. Second, they provide the user a convenient way to store a description of a complex component, so that data about it do not need to be entered each time the component is named in a circuit description. Instead, the CDL portion of ATG uses the Analog Component Library element to expand the component into its constituents, so that ATG can write tests for the individual devices within it.

The third use of ACL elements is to link non-primitive components to tests with the same type names in the Analog Test Library. This allows the user to enter non-primitive components in a circuit description and use tests for them that have been provided in the Analog Test Library. Refer also to para 2.4 for the role of the ACL in test generation.

A device is entered in the Analog Component Library under a descriptive name, which the test generator regards as its type designation. Thereafter the device can be entered in a circuit description file just as a primitive type (e.g, R, C, CR, etc.) is entered. If the component is given an alias in the library, the alias can also be used as its type. A number of common complex devices are provided in the system ACL; the user can create new analog component descriptions by creating entries in a User ACL. Refer to para 4.5 for instructions on the use of the ACL Monitor Options Page.

During test generation, ATG will search the Analog Component Library for component descriptions; then it searches the Analog Test Library for corresponeing tests. If a device is described in the Analog Component Library, ATG writes a test for each individual component in the device description. If an entry for the same device appears in the Analog Test Library, ATG will then also write a test for the device as a whole, i.e., a functional test of the device.

Each ACL component description consists of one or more primitive devices described as they would be described in a circuit description file. An ACL component can also contain complex components described elsewhere in the ACL, but complex components can be nested in this way only to one level. That is, a complex component may only contain another complex component that consists solely of primitive devices.

The CDL task inserts connection information about these primitive devices into the circuit description work file (.WOR) while maintaining the identity of the complex component for generating tests, reports, and error messages. A 3-file library structure exists with System and Update libraries provided by GenRad; users may create their own User .ACL files. The System file contains a small number of popular devices both used by ATG and available as models for customer-written component descriptions.

7.2 WRITING A COMPONENT DESCRIPTION

A component description file (.ACS) is written like a circuit description file, using the internal schematic of the component. All component pins must be coded in the .ACS model as correspondingly named EXTernals. The .ACS format and language are identical to those of a .CKT, except that the .ACS contains no %ADAPTOR section. After being written, the .ACS file is inserted into the User ACL using procedures described in paragraphs 3.2 and 3.3 of this manual.

A complex component model must consist of ATG primitive devices and have no inaccessible internal nodes (i.e., each device must have all its terminals connected to component pins). Every node must be connected to one and only one EXTernal, named with the corresponding component pin name or number. A device with a number of external pins connected to a single internal node can be modeled by creating a number of dummy nodes in the device (one node for each pin). These nodes are then connected together, in the library description, with jumpers.

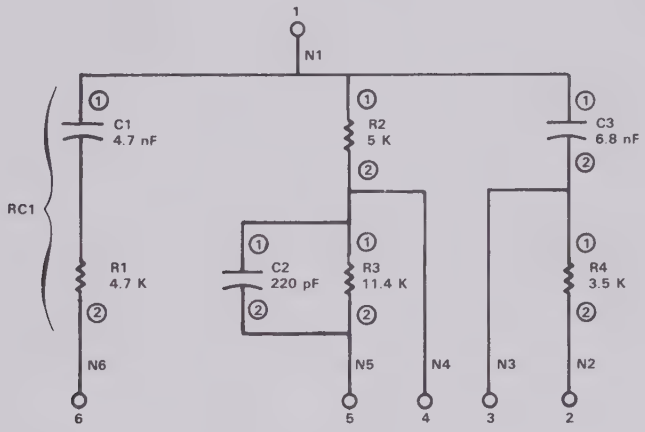
Complex devices can have inaccessible internal nodes; the ACL model must be written in such a way, however, as to show no inaccessible internal nodes. If it is necessary to describe a group of devices and nodes that cannot be tested, or that have unknown characteristics, lump them as one device and enter them as device UKnn with a value of =NOTEST, as shown for RC1 in Figure 7-1. ATG will not write tests for such a device, and it will not include the device when it evaluates tests for other components on the board which are connected to them. Do not use unconnected EXTernals to represent inaccessible nodes.

Section 6 of the Test Set Preparation manual contains information on writing %CIRCUIT and %VALUE sections for both .CKT and .ACS files. Each basic device in the component is listed with its type and nodal connections in the %CIRCUIT section of the library description; its value, tolerance, and any applicable flagspecs are given in the %VALUE section. Components can usually have pin names consisting of up to 4 alphanumeric characters. However, some complex analog components may also be listed in the Digital Test Library; such components must have pin numbers (not names) and must be named with the same type designation used in the DTL.

Once contained in a library, complex components can be entered in a circuit description (.CKT) file in the same way as any primitive component, such as a resistor or transistor. The name or alias of an ACL library element is used as a complex component type, just as R is used as a resistor type. When the circuit description file containing the ACL component is compiled, data from the .CKT file are merged with data from the compiled .ACS file to provide a complete working file (.WOR).

An example of a component description file, in this case for an RC network, is shown in Figure 7-1.

The example in Figure 7-1 is fairly straightforward; the basic devices in the RC network component are listed as they would be in a circuit description file. However, R1 and C1 are not testable because the node between them is purely internal to the component and not accessible to the test system. For this reason they are entered as a single device RC1 of a type UK2. This device-type alerts ATG that RC1 is an unknown 2-terminal device that cannot be tested. Entering RC1 as a UKnn device makes entering avalue for it in the %VALUE section unnecessary.




```
/* RC NETWORK ASSEMBLY */

%CIRCUIT;

/* NAME        TYPE     CONNECTIONS */

   1           EXT      ,N1 RC1.1 R2.1 C3.1;
   2           EXT      ,N2 R4.2;
   3           EXT      ,N3 C3.2 R4.1;
   4           EXT      ,N4 R2.2 C2.1 R3.1;
   5           EXT      ,N5 C2.2 R3.2;
   6           EXT      ,N6 RC1.2;
   RC1         UK2       N1 R2.1, N6;
   R2          R         N1 C3.1, N4 C2.1;
   R3          R         N4 R2.2, N5 C2.2;
   R4          R         N3 C3.2, N2;
   C2          C         N4 R3.1, N5 R3.2;
   C3          C         N1 RC1.1, N3;

%VALUE:2270;

/* COMPONENT   VALUE     TOLERANCE  FLAGSPECS */

   RC1         =NOTEST;
   R2          =5K,      5%;
   R3          =11.4K,   5%;
   R4          =3.5K,    5%;
   C2          =220P,    10%;
   C3          =6.8N,    20%;
```

Figure 7-1. Component Description Example.

System Library Tests – Section 8

8.1 GENERAL

The test library contains a description of all characteristic tests evaluated and used by ATG in developing test programs. In the process of developing the test program, ATG refers to the test library for both the overall testing sequence and for the individual test configurations. For each component on the board, ATG refers to the appropriate test descriptions in the library, selects an acceptable one, and develops the test program segment that will execute that test.

The test library actually consists of 6 library files: the System Analog Test Library, the Update Analog Test Library, the User Analog Test Library, the System Digital Test Library, the Update Digital Test Library, and the User Digital Test Library. The capability for the user to list and to modify analog and digital tests is provided. The Test Library Programming manual gives a detailed description of the ATG test library, including the procedure for creating and maintaining the libraries.

The Analog Test Library contains all the utility routines for setting up the test system. It also contains all the routines which do connectivity testing of a circuit board. Therefore, it is necessary to process the Analog Test Library for all automatic test generation even when no analog components are on the board. Tests exist in the Analog Test Library for each analog component type and in the Digital Test Library for each IC type.

8.2 ATG TESTS

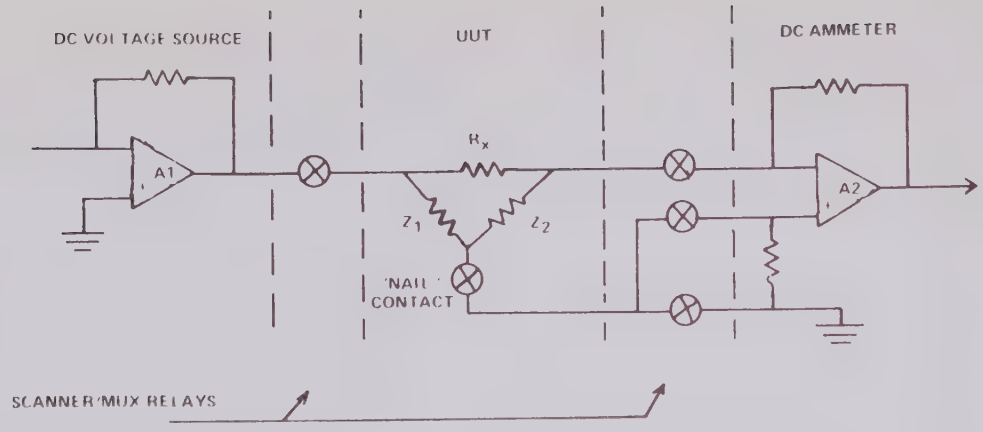
8.2.1 General

This section presents a general description of the types of tests contained in the ATG test library. It briefly describes the method of testing, the various measurement configurations, and the selection criteria used by ATG to determine the acceptability of these tests. The actual test description coding can be obtained by listing the test library.

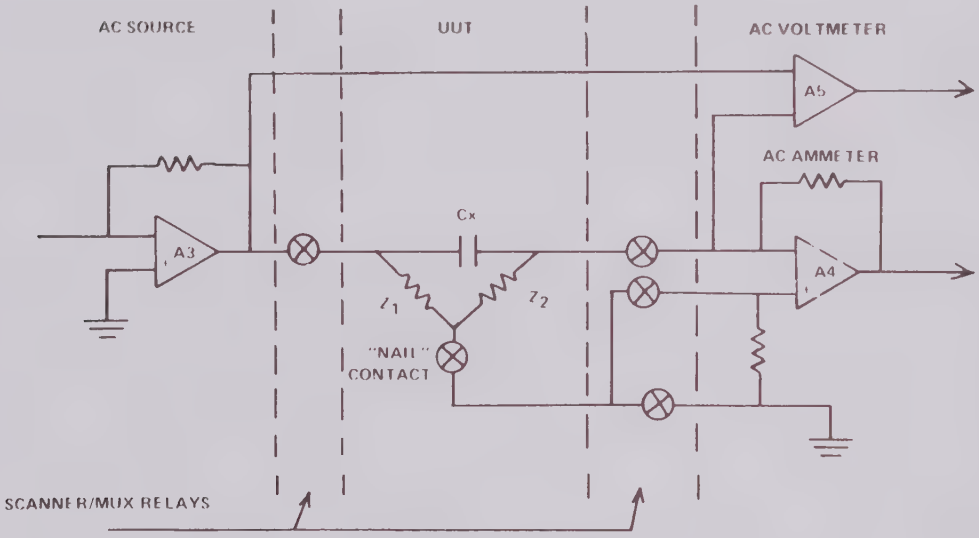
8.2.2 Types of Measurements

Analog

There are 2 basic types of test configurations used in testing most passive components: Guarded tests and Non-guarded tests. A description of these tests is given in para 1.4 of the System Operation manual. The diagrams for guarded tests are repeated in Figure 8-1 for reference. The 2 scanner contacts connected to the nail on the guard node are for the purpose of reducing errors introduced by scanner contact resistance at the guard node. This is accomplished by connecting the instrument voltage reference to the scanner output rather than directly to ground. Unique applications of guarding techniques may be used in the test configurations of active components. Detailed descriptions of all analog library tests are included later in this section.



Guarded Resistance Measurement.



Guarded Reactance Measurement.

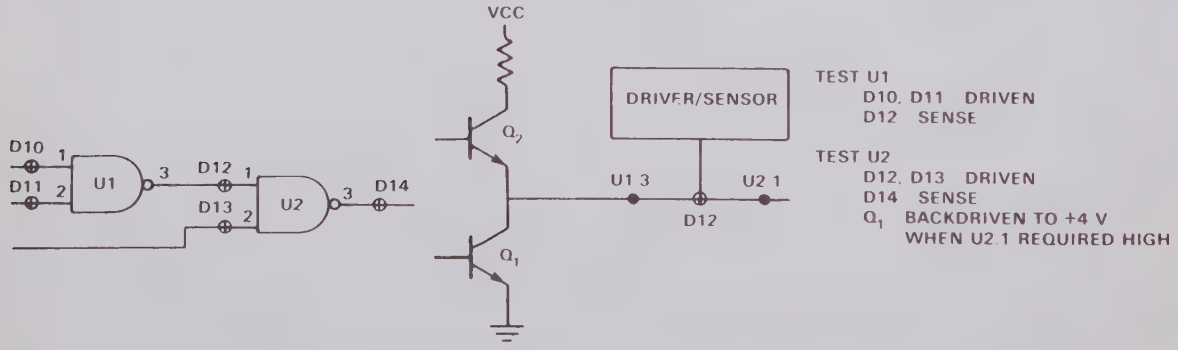


Figure 8-1. Typical In-Circuit Measurement Connections.

Within each of the basic test types there may be several test configuration variations derived from combinations of the following:

Applying the source to different leads of the component

Varying the source magnitude

Changing the guard connections with respect to source and measure nodes

ATG evaluates the various test configurations provided in the library until it accepts a test. Guarded test configurations are usually evaluated before non-guarded test configurations. If ATG cannot find an acceptable test, it inserts the best test found, following a branch instruction that prevents the test from being executed. A non-executing test of this type will be identified in the report file (<board name>.RPT) as a FORM test. It should be evaluated and modified by a test engineer before the branch instruction is removed to incorporate the test into the program.

Digital

Digital devices are tested by applying a series of logic levels to the inputs of the device and monitoring the outputs. The test library contains one test procedure for each device type. This procedure is written so that ATG customizes it for a particular circuit connection pattern. Some of the considerations that may be made by digital ATG in generating a test for a device are suggested by the following:

What gate inputs are tied together by the circuit and cannot be driven by different states?

What gates are tied to a bus and must be disabled while other gates are being tested?

What pins must be kept isolated from each other?

Is there a bus fault and what device is causing the fault?

What devices are on a bus with a fault and thus cannot be tested?

The digital ATG process consists of two tasks. The pre-processor task (CDE) creates two data files, .DAI and .IDD or .IDX. The .DAI file is used by the second task, the Digital Test Generator (DTG). The .IDD file is also used by DTG and is saved for diagnostic use during run-time of an actual board test. The circuit information in these files is arranged so that DTG can use it for making the various test generation decisions.

8.2.3 Selection Process for Analog Test Procedures

The analog test library determines the order in which the various test configurations are evaluated. A component type will be represented in the library by one or more basic tests. Within each of these basic tests there may be several measurement configurations. The basic tests are evaluated in the order that they are listed in the library directory. The order of evaluating the various measurements within a basic test is determined by the order in which they are written into the library routine. The measurement configurations which are generally most likely to be acceptable are written first. ATG will sequentially evaluate the measurement configurations until it finds one where

all circuit conditions are favorable and all measurements can be made within the defined error limits. As soon as this happens, ATG will write the test procedure into the test program and then proceed to the evaluation of tests for another component. ATG does not search for any more configurations once an acceptable test is found.

When ATG evaluates a measurement configuration and finds that it does not meet all the specified requirements, it decides whether or not the test procedure is good enough to save for later consideration for inclusion into the test program. If a test procedure had been saved from the evaluation of a previous configuration, the best of the two will be saved for further consideration. If all configurations are evaluated with no test having met all the requirements, the last test procedure saved, if any were saved, will be written into the test program.

In the case that all configurations are evaluated with no test procedure being adequate for inclusion in the test program, ATG writes a FORM test procedure. A branch instruction to the statement following the procedure is placed ahead of the procedure in the test program to prevent the test procedure from executing. After making appropriate modifications, the user can choose to remove the branch instruction to allow the test procedure to be executed.

Measurement Error Limits

The user has some control over the measurement error limits that ATG uses in making one of the following decisions:

1. Write this test and stop looking. For purposes of discussion, call the measurement limit for this decision the write limit (WRILIM).
2. Save this test and continue looking. For purposes of discussion, call the measurement limit for this decision the save limit (SAVLIM).
3. Ignore this test and continue looking.

There are 3 measurement error flagspecs available for inclusion by the user in the circuit description which effect the measurement limits of the passive components R, L and C. These are RERR, LERR, and CERR. These flagspecs are called XERR for this general discussion. ATG assumes a value of 1.0 for XERR if a value is not specified in the circuit description. ATG calculates WRILIM to be the sum of the component nominal value (VAL), the component tolerance value (TOL), and the test measurement error. The test measurement error is TOL multiplied by XERR. If TOL is not symmetrical about VAL, then the larger magnitude is used in the calculation of test measurement error. Test measurement error is a parameter which represents the allowable measurement error caused by instrumentation accuracy and by circuit conditions around the component to be tested. The write measurement error limit is defined by

$$WRILIM = VAL \pm (TOL + (XERR \times TOL))$$

The save measurement error limit is defined by

$$SAVLIM = VAL \pm VAL$$

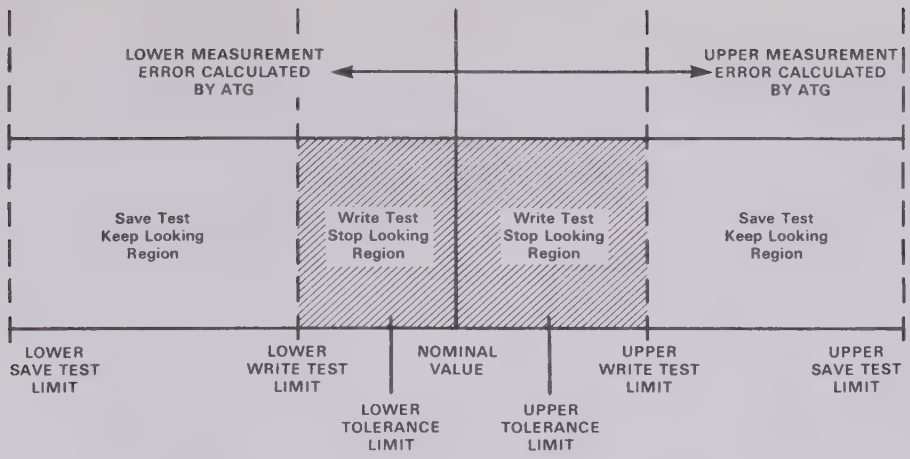

LOWER MEASUREMENT ERROR CALCULATED BY ATG
UPPER MEASUREMENT ERROR CALCULATED BY ATG
Save Test Keep Looking Region
Write Test Stop Looking Region
Write Test Stop Looking Region
Save Test Keep Looking Region
LOWER SAVE TEST LIMIT
LOWER WRITE TEST LIMIT
NOMINAL VALUE
UPPER WRITE TEST LIMIT
UPPER SAVE TEST LIMIT
LOWER TOLERANCE LIMIT
UPPER TOLERANCE LIMIT

Figure 8-2. ATG Test Selection Limits.

Example

Assume a 1-kilohm resistor with a tolerance of ±5% for which the user decides that the test measurement error should be 80% of the tolerance.

RERR = 0.8

WRILIM = 1000±(1000 x .05 +(0.8 x 1000 x.05))
= 1000 ± (50 + 40)
= 1090 ohms and 910 ohms

SAVLIM = 1000 ±1000
= 0 ohms and 2000 ohms

It may be noted that the smaller the test measurement error, the more likely that test configurations will fail. This means that ATG will have to try more configurations and thus the ATG process will take longer to run. The worst-case spread in time for a single component could be from a few seconds if the first configuration is accepted to approximately 10 minutes if all configurations must be tried.

The selection criteria discussed in the preceding paragraphs are illustrated in Figure 8-2.

The ATG process analyzes all the factors which contribute to test measurement error. An error value is computed for each factor. The sum of all these error values becomes the figure that ATG compares to the Write Test Limits and the Save Test Limits. The actual measurement errors calculated by ATG become the test limits used in the final test procedure for the component.

Circuit Condition Considerations

There are a number of conditions in addition to the measurement error which may cause a test not to be selected. These conditions are determined to exist by two means: one by decisions that are programmed into the library routines, the other by decisions which are programmed into the ATG process routines. The library decisions result from conditional QUIT and QUIT IGNORE commands in the library test files. The ATG process decisions are executed by VETO commands.

The QUIT decision is made when a condition is found that indicates the type of test being evaluated is not suitable. It causes ATG to stop looking for a test configuration of that type and to look for a configuration of another type for this component. If there is no other type of test to be evaluated, a FORM (non-executing) test is written.

The QUIT IGNORE decision is made when a condition is found that indicates that the component cannot be tested. It causes ATG to stop looking for a test configuration for this component and to go to the generation of a test for the next component.

The VETO decision is made when ATG determines that a condition exists which may be harmful to a component in the circuit to be tested or which may cause ambiguous results. Some of the several VETOs that may be invoked are:

veto 2 when the back voltage on a CP1 capacitor is less than zero volts

veto 5 when there is no nail on the node that needs to be guarded

veto 6 when a non-linear device in the circuit will cause clipping of the source voltage and thus provide an inaccurate measurement

veto 10 when an applied voltage might damage a device on the board, or

when an applied voltage is specified outside the limits of ATG, or

when the back voltage on a CP capacitor is less than -1.01 volts, or

when there is a high voltage near an IC while a passive device is being tested.

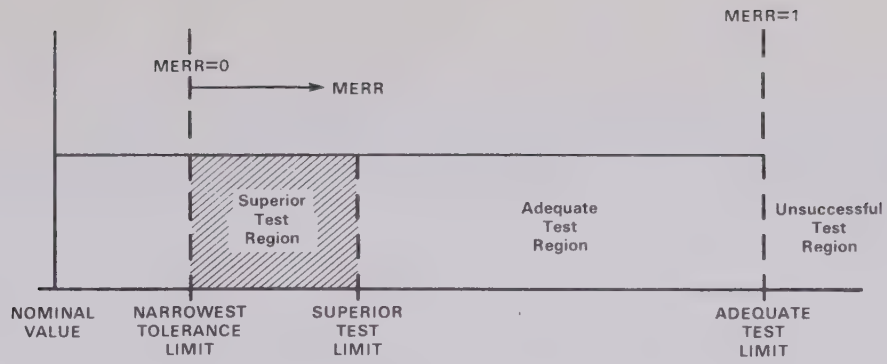
VETOs inhibit the use of the test configuration in which the VETO occurred whenever the current veto level (normally set at 4) is less than the VETO condition. ATG would continue looking for an acceptable configuration.

However, when the SINgle component name is set on the ATG Monitor Options page, normally vetoed tests will be accepted. This allows ATG to write a test that can be analyzed by the library programmer. The SINgle option is usually used when test programs are being developed, and the veto would hinder debugging of test procedures.

8.2.4 Classification of Selected Analog Tests

Once ATG has selected a test for a component, it does a check to determine whether to classify it as a "superior test" or an "adequate test." This classification does not effect the test procedure; it only determines the statistic that is put into the report file (.RPT). ATG takes the larger of the upper and lower test measurement errors that were calculated in the selection process and calls this value MERR. MERR is compared with a specified error limit. The error limit is controllable by the RPT flagspec that the user can specify in the circuit description. This flagspec is used by ATG to determine the breakpoint between a "superior" and an "adequate" test in accordance with the following formula:

$$\text{Superior Test Limit} = \text{NTOL} \pm (\text{NTOL} \times \text{RPT})$$



NOTES: 1. The adequate test limit is + or − 100% of the nominal value.
2. MERR = 0 is the zero measurement error value.
3. MERR = 1 is the 100% measurement error value.
4. The narrowest tolerance limit is + or − the smaller tolerance value in non-symmetrical component tolerance spec.

Figure 8-3. ATG Test Classification Limits.

ATG will assume a value of 1 for RPT if not specified. The narrower of the upper and lower tolerance values of a non-symmetrical tolerance (NTOL) is used except when the narrower value is zero. The other value is then used. The adequate test limit is:

$$\text{Adequate Test Limit} = \text{VALUE} \pm \text{VALUE (or 0 to 2 x VALUE)}$$

It may be noted that the Adequate Test Limit is equal to the SAVE TEST Limit in Figure 8-2. The superior test limit is equal to the Write Test Limit if the flagspecs XERR and RPT are both 1. Figure 8-3 illustrates Test classification. MERR will never exceed the adequate test limit because MERR is only calculated when a test is written. No test is written if measurement error exceeds the adequate test limit. In the case where no test is written, the selection process reverts to the writing of a FORM test. When a FORM test is written, the statistic "Unsuccessful test" is added to the report file.

A "Not Testable" statistic is added to the report file when ATG determines that a meaningful test is not possible, for example, a resistor paralleled by a jumper.

8.2.5 Selection Process for Digital Test Procedures

Test selection or test structuring for digital IC test procedures consists of analyzing the basic test procedure that is in the library. ATG attempts to match each pin of the IC to a circuit node and then decides whether the pin can be tested. Gate inputs tied together would mean that a test BURST would have to delete the steps that required these two inputs to assume different states. Unused connections would not be included in a test.

If the gates are specified to be disabled or inhibited for isolation from other pins, ATG will check to be sure it is able to perform this isolation and will report the results if the isolation cannot be completed successfully. It is not always possible to be sure that the ATG process was successful in disabling all IC outputs that are connected to a bus. Therefore, ATG automatically provides a bus test in the digital test program to check to see if the bus is free prior to doing any bused IC tests. ATG organizes the digital test program so that all non-bused ICs are tested before the bused IC tests. Therefore, if the bus test fails, the non-bused ICs will be tested but the bused ICs will not.

Many ICs have separate, identical circuits which are treated as equivalent sets in the library procedure. ATG rewrites the set procedure for each circuit, inserting the correct pins each time. Any equivalent sets that are not connected in the circuit are disabled in the generated procedure.

The system provides for checking IC pin connections to the circuit board when the test for that IC has failed. Many times, an IC test failure is due to connections, and it is important to check for this before condemning the IC. If the SCRATCHPROBE option on the DIAGNOSE page is enabled, the system will automatically display the failed IC on the video display and instruct the operator to use the Scratchprobing diagnostic technique to probe the IC pins. Refer to para 3.3.3 in the System Operation manual for more details about using the probe. The result of this Scratchprobing technique will determine the type of failure reported.

The automatic probing takes place on any digital IC that is included in the digital test library. The probe is connected to a digital pin in the adaptor. This pin must be specified for the Scratchprobing technique to be executed. It is specified in one of two ways. One way is to specify it in the ADAPTOR section of the circuit description file (.CKT). Refer to para 7.2.6 in the System Operation manual. The other way is by defining the pin on the diagnose sub-monitor option page by filling in the "Number of probe nail =" option. Refer to para 3.3.3 in the System Operation manual. If the pin is specified in the .CKT file, the option can be "0;" however, if it is also specified on the options page, the pin entered on the options page will take precedence if there is a conflict.

8.2.6 Utility Test Procedures

The first step performed by a test program is setting up the equipment. This involves opening channels for I/O devices, declaring variables, and declaring subroutines. The last step is the orderly closing of I/O device channels. All the test language statements that accomplish these steps are contained in the Analog Test Library and are written into the test program whenever ATG is run. The library routine for doing this is called a utility.

8.2.7 Connectivity Test Procedures

Two connectivity tests are included at the beginning of every ATG-produced test program. Although these tests exist only in the Analog Test Library, they are automatically included in any test program, even if there are only digital components.

First, an opens test checks all external connections. In the case where more than one adaptor pin is connected to the same electrical node (e.g., several adaptor pins extended over an etched track in the pc board), several continuity tests are performed (one less than the number of pins) to ensure that the pins contact the node.

Secondly, a shorts test performs an "intelligent" search for shorts between circuit nodes. The search algorithm is designed to check for shorts efficiently while still keeping the test within reasonable limits. This search takes into account apparent shorts caused by inductors, jumpers and fuses. Momentary shorts caused by uncharged capacitors are considered by the 'run time' software rather than by ATG. Also, if a resistor connected between two circuit nodes has a specified value of less than the threshold level, then the test

considers those two nodes as a single point and no short is reported. For example, if the threshold level is 10 ohms, a 10-ohm, 5% resistor or a 12-ohm, 20% resistor would not cause a short to be reported. If the specified value and tolerance of the resistor is greater than the threshold (e.g., 12 ohms, 10%), the two nodes are treated as separate points.

Unless otherwise specified, the threshold level for analog connectivity tests will default to 10 ohms. The digital threshold is 40 ±30 ohms. The analog threshold value can be changed, however, by including the following statement type in the %VALUE specification:

```
BOARD = SHORT = 20;
```

ATG will accept threshold values from 5 to 1000 ohms. If the value is specified outside these limits, ATG sets the threshold to the default values without a message being produced. The digital tests will ignore this flagspec. If the circuit contains any CP1 devices, shorts testing is performed at 50 mV. Otherwise, testing is performed at 200 mV.

8.2.8 Resistance Test Procedures

Resistance is measured by applying a dc voltage to the component under test and measuring the resultant current. The ATG Test Library provides two basic test configurations with several measurement variations in each. The two basic measurement configurations are non-guarded and guarded.

The library procedures are arranged so that ATG will take a minimum average time in searching for a test. Guarded tests are analyzed first, because ATG is more likely to find a "superior" test among them. The one exception to this is when the circuit description indicates that a resistor is connected to an EXTernal point. There is no need to guard a resistor that has one end open, so ATG goes immediately to unguarded tests for this case. The configurations within guarded and unguarded test groupings are ordered so that the most likely ones are tried first. Generally, the lower voltage configurations are tried before the higher voltage configurations, the configuration with the source applied to the node with the most connections is tried before the ones with the source applied to the node with the least connections, and the configuration where the measure node is guarded is tried before the configuration where the source node is guarded.

Table 8-1 lists all the guarded configurations in the order in which they are generally tried by ATG. The unguarded configurations all have the source on the node with the most connections and consist of the following source voltages: +0.2 V, -0.2 V, +1.0 V, -1.0 V, +5.0 V, and -5.0 V.

The value of resistance has an effect on what voltage will be most likely to generate a test. Low resistance measurements would not likely find a safe test with high source voltage. High resistance measurements would not likely find "superior tests" at the low source voltage. ATG will automatically start evaluating tests at the 5-volt level when the value of resistance is greater than 1 megohm. It will do only the 0.2-volt level when the value is less than 500 ohms. Otherwise, ATG will start at the 0.2-volt configurations and proceed to the higher voltages if necessary. These values of resistance may be changed by the user by specifying values for the RHIV and RLOV flagspecs in the .CKT file. Table 8-1 shows where ATG starts searching as a result of the flagspecs.

ATG may be sped up even more by the user's specifying FASTATG in the circuit description. When this is done, ATG only looks at the configurations indicated in Table 8-1. The trade-off for gaining ATG time is generally a reduction in the percentage of "superior" tests found.

Table 8-1
GUARDED MEASUREMENT CONFIGURATIONS

Source VDC	Source node at component pin with	Node Guarded	Effect of flagspec on starting configuration [default]
+0.2 V	most connections	measure	FASTATG*
+0.2 V	least connections	measure	FASTATG*
+0.2 V	most connections	source	
+0.2 V	least connections	source	
-0.2 V	most connections	measure	
-0.2 V	least connections	measure	
-0.2 V	most connections	source	
-0.2 V	least connections	source	
+1.0 V	most connections	measure	
+1.0 V	least connections	measure	
+1.0 V	most connections	source	
+1.0 V	least connections	source	
-1.0 V	most connections	measure	
-1.0 V	least connections	measure	
-1.0 V	most connections	source	
-1.0 V	least connections	source	
+5.0 V	most connections	source	R>RHIV [1 MEG]
+5.0 V	least connections	source	
-5.0 V	most connections	source	
-5.0 V	least connections	source	

* Only these two tests can be used with FASTATG.

Components that ATG can evaluate for resistance are resistors, inductors and fuses. Test evaluations for these devices may be controlled by flagspecs in the circuit description.

Flagspecs available for user control of resistor test selection are:

RERR	DCR
RHIRESID	RTOL
RLORESID	RLOV
ILIM	RHIV
FASTATG	

These flagspecs are defined in the following paragraphs:

RERR - Used to set error limits that determine if ATG accepts the test which it just evaluated and stops looking for a better test. The values that are specified are treated as an add-on factor to the component tolerance. If RERR is specified as 0.85 and the resistor tolerance is 10%, the allowable test error is 8.5% of the resistor value and the pass limit is 18.5% of the resistor value. The default value for RERR is 1.0.

RHIRESID - There is a certain amount of residual resistance in the
RLORESID system due to mux, scanner, and fixture cabling. This resistance can introduce a significant error when the component being tested has a small resistance value. It is also evident that this residual resistance will vary from one connection to the next. ATG includes a scheme that compensates for this variable resistance. The user can determine a high and a low value of residual resistance for each applicable component. These measured values are inserted into the Circuit Description Language %VALUE section as flagspecs, named RHIRESID and RLORESID. The default values for these are RHIRESID=3.2 ohms and RLORESID=1.8 ohms. ATG calculates a nominal value of residual resistance and adds it to the component test limits. It treats the variation in residual resistance as measurement error.

ILIM - Used to specify the limiting value of source current allowable during a test. The default value is 0.128 amp. The only other value that can be selected with the standard voltage source is 0.020 amp.

RTOL - This flagspec specifies the tolerance of the dc resistance of an inductor. There is no default for this flagspec. It must be specified for an inductor or no resistance test will be generated.

DCR - This flagspec specifies the dc resistance value of an inductor. There is a default value of 1.0 ohm; however, this default value is only for determining circuit impedances for other tests. If a dc resistance test is desired for an inductor, both DCR and RTOL must be specified.

RLOV - This flagspec is used to save ATG processing time for devices with low resistance values. ATG will bypass all library test configurations that would use a source voltage greater than 0.2 volt when the device resistance is less than or equal to the value specified in this flagspec. The default value is 500 ohms.

RHIV - This flagspec is used to provide a more accurate test of devices with a high resistance. ATG will go directly to a guarded test configuration that uses a 5-volt source when the component resistance is greater than the value specified in this flagspec. The default value is 1.0 megohm.

FASTATG - Causes ATG to evaluate only test configurations at +0.2 V source voltage to save ATG process time. The effects of selecting FASTATG are: less accurate tests of resistors with high value because of low source voltage; less likelihood of finding a test because all source and guard configurations are not tried; and no test generated when there are CP1 capacitors connected so that a reverse bias would exist on them.

8.2.9 Potentiometer Test Procedures

The potentiometer (pot) test is similar to the fixed resistance test, except a definition of the pot setting must be determined. The method of specifying this setting is explained below. If the resistor is not pre-set to a known setting, the test interacts with the user to establish one.

Potentiometer adjustments are always performed near the beginning of the test program, immediately following the connectivity test and before the board shorts tests. The system will determine if the pot is preset to the specified or the default position. It will bypass the interaction with the user when it is properly set.

Flagspecs

The total pot resistance must be specified in the circuit description. Two flagspecs, in addition to all those used with the fixed resistance test, are available for defining the pot setting as follows:

R23 Resistance between the wiper (pin 2) and the clockwise end of the of the pot (pin 3). Normal pot convention provides for the wiper (pin 2) to move from pin 1 to pin 3 when turned clockwise. Therefore, this flagspec will always cause the resistance between pin 1 and the wiper to increase when the pot is adjusted in the clockwise direction.

PRESET Keyword indicating to ATG that the pot is pre-set to a known setting.

Depending on what flagspecs are specified, the following pot tests are performed:

1. No flagspecs defined For example,

```
RV7 = 10K, 20%;
```

In this case, the resulting test program will guide the user (via messages on the display) in adjusting the pot to its mid-point value if the pot is used as a divider. If the pot is a variable resistor (wiper tied to one end), the displayed messages will guide the user to adjust the pot to its maximum value.

2. Flagspec R23 defined and PRESET defined For example,

```
RV7 = 10K, 20%, R23=1K, PRESET;
```

This causes ATG to test the pot as two fixed resistors (e.g., 1K and 9K). The pot must be preset to the specified setting.

3. Flagspecs R23 defined (Adj.) For example,

```
RV7 = 10K, 20%, R23=1K;
```

This will cause the resulting test program to guide the user in adjusting the pot to the value specified by R23.

8.2.10 Capacitance Test

Capacitance measurements are made by the ACZ unit. This unit applies an ac source voltage to the source node of the component-under-test and measures the in-phase and quadrature components of both the voltage across the component and the current through it. The system software then uses these measured values to calculate the actual capacitance.

ATG evaluates the different test configurations as follows:

1. If the capacitance to be measured (CX) falls outside the range of 15 pF < CX < 2000 uF, ATG reports the expected measurement error due to the out-of-range value, but the test selection process continues.

2. If the shunt susceptance in parallel with CX is equal to, or greater than, 100% of the susceptance of CX, a "NO TEST" condition is reported and the test selection process terminated.

3. If this shunt susceptance is greater than 1% but less than 100% of the susceptance of CX, the pass limits for that test are adjusted and a message generated indicating that test results will include this parallel susceptance.

4. User-selectable frequencies for non-polarized capacitors are 100 Hz, 1 kHz, or 10 kHz (±1%) and, for polarized capacitors, 100 Hz only. If any other frequency is specified, a "NO TEST" condition exists. Unless otherwise specified by the user, the source frequency (100 Hz, 1 kHz, or 10 kHz) is selected to minimize instrument error. This error is dependent on the value of the source voltage and the capacitance being measured.

If ATG does not find an acceptable test, then a "form" test with all the best parameters, or a note that guides the user in selecting a parameter, is inserted in the test program listing.

There are two types of capacitors that may be tested: polarized and non-polarized. The ATG Test Library contains guarded and non-guarded tests for each type. The various measurements for each of the configurations are listed below.

Non-Polarized Capacitor (C)

There are 4 unguarded measurement configurations. They all have the source connected to the node with the most connections. The source voltage and bias voltage are as follows:

Source (Vrms)	VDC Bias	Comments
1.0	0	
0.2	0	only configuration tried when FASTATG is specified
0.2	+0.3	tried only if CP1s on board
0.2	-0.3	tried only if CP1s on board

There are 16 guarded measurement configurations with the following parameters.

Source (Vrms)	VDC Bias	Source Node	Node Guarded	Comments
1.0	0	most conn.	source	
1.0	0	most conn.	measure	
1.0	0	least conn.	source	
1.0	0	least conn.	measure	
0.2	0	most conn.	source	only 2 configs. tried
0.2	0	most conn.	measure	when FASTATG spec.
0.2	0	least conn.	source	
0.2	0	least conn.	measure	
0.2	+0.3	most conn.	source	
0.2	+0.3	most conn.	measure	
0.2	+0.3	least conn.	source	these tried only if
0.2	+0.3	least conn.	measure	CP1s on board
0.2	-0.3	most conn.	source	
0.2	-0.3	most conn.	measure	
0.2	-0.3	least conn.	source	
0.2	-0.3	least conn.	measure	

Polarized Capacitor (CP or CP1)

A polarized capacitor is designated CP1 in the CDL only when it is a type which cannot tolerate any reverse bias. All others are designated CP, in which case ATG will accept any measurement that does not exceed 1.0-V reverse bias.

There are 5 unguarded configurations. They all have the source connected to the node with the most connections. The source and bias voltages are as follows:

Source (Vrms)	VDC Bias	Source Node	Comments
1.0	+1.5	Pos. lead	Skip first 2 config.
1.0	-1.5	Neg. lead	if FASTATG spec.
0.2	+0.3	Pos. lead	Not tried if CP
0.2	-0.3	Neg. lead	Not tried if CP
0.2	0		Not tried if CP1

Twenty guarded measurement configurations are available for CPs and CP1s:

Source (Vrms)	VDC Bias	Source Node	Guard Node	Comments
1.0	+1.5	most conn. Pos. lead	source	
1.0	-1.5	most conn. Neg. lead	source	
1.0	+1.5	most conn. Pos. lead	measure	

Source (Vrms)	VDC Bias	Source Node	Guard Node	Comments
1.0	-1.5	most conn. Neg lead	measure	
1.0	+1.5	least conn. Pos. lead	source	
1.0	-1.5	least conn. Neg. lead	source	
1.0	+1.5	least conn. Pos. lead	measure	
1.0	-1.5	least conn. Neg. lead	measure	
0.2	+0.3	most conn. Pos. lead	source	CP1 only, FASTATG
0.2	-0.3	most conn. Neg. lead	source	CP1 only, FASTATG
0.2	+0.3	most conn. Pos. lead	measure	CP1 only, FASTATG
0.2	-0.3	most conn. Neg. lead	meas.	CP1 only, FASTATG
0.2	+0.3	least conn. Pos. lead	source	CP1 only
0.2	-0.3	least conn. Neg. lead	source	CP1 only
0.2	+0.3	least conn. Pos. lead	measure	CP1 only
0.2	-0.3	least conn. Neg. lead	measure	CP1 only
0.2	0	most conn.	source	CP only, FASTATG
0.2	0	most conn.	measure	CP only, FASTATG
0.2	0	least conn.	source	CP only
0.2	0	least conn.	measure	CP only

Flagspecs

Flagspecs available for user control of capacitor test selection are:

CERR
FREQ
CHIRESID
CLORESID
FASTATG

CERR — This flagspec is used in the same manner as RERR; default is 1.0.

FREQ — Selects the source test frequency for non-polarized capacitors from 100 Hz, 1 kHz, or 10 kHz.

CHIRESID
CLORESID — These flagspecs are used in the same manner as the resistance residual flagspecs. Default values are CHIRESID = 20 pF and CLORESID = 5 pF.

FASTATG — Causes ATG to execute in less time by skipping some configurations as noted in the preceding list.

8.2.11 Inductance Test Procedures

The ATG Test Library contains inductance measurements for non-guarded and guarded configurations. Inductance measurements are made by the ACZ unit. As in a capacitance test, the ACZ applies an ac voltage to the source node of the component-under-test and measures the resulting complex voltage across the component and the current through it. A dc resistance test is also done on inductors prior to performing the inductance measurement. The DCR and RTOL flagspecs must be specified or the librarian will not allow the component test to be inserted in the .ACS. Refer to the DCR flagspec in para 8.2.8 for details of inductor resistance tests.

There are 2 unguarded measurement configurations. One uses a 1.0 Vrms source and the other uses a 0.2 Vrms source. Both have the source applied to the node with the most connections. Only the 0.2 Vrms configuration is used when FASTATG is specified in the circuit description.

The guarded measurement configurations that are available are:

Source (Vrms)	Source Node	Guard Node	Comments
1.0	most conn.	source	
1.0	most conn.	measure	
0.2	most conn.	source	FASTATG
0.2	most conn.	measure	FASTATG
1.0	least conn.	source	
1.0	least conn.	measure	
0.2	least conn.	source	
0.2	least conn.	measure	

Flagspecs

Flagspecs available for user control of inductance test selection are:

LERR LLORESID
FREQ FASTATG
LHIRESID

LERR — This flagspec is used in the same manner as RERR; default is 1.0.

FREQ — Selects the source test frequency. If it is not specified in the Circuit Description, the ATG chooses a frequency that optimizes for minimum instrument error and is based on the expected value of inductance. This is done according to the following list:

Inductance Range	Source Frequency
Less than 5.1mH	10 kHz
5.1mH to 2.6H	1 kHz
More than 2.6H	100 Hz

LHIRESID, LLORESID — These flagspecs are used in the same manner as the resistance residual flagspecs. Default values are LHIRESID = 158uH and LLORESID = 5uH.

FASTATG — Causes ATG to execute in less time by only trying fewer configurations as indicated above.

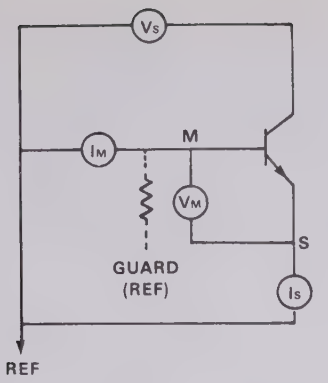


Figure 8-4. Transistor Test Connections.

8.2.12 Transistor Test

The transistor tests are structured around the instrument configuration shown in Figure 8-4. The ATG Test Library contains two tests for each transistor type (NPN and PNP). One of these tests (SIMPLE) is used where the transistor has no direct shunt elements across any of its three junctions. The other test (DELUXE) is used where there are direct shunt elements in the circuit. In either case, all equivalent resistances from collector-to-ground, base-to-ground, and emitter-to-ground are considered in the test evaluation.

In evaluating the various test configurations, ATG checks each to ensure that:

1. The total current through the transistor and adjacent resistors does not overload the voltage source. If it does, the collector voltage is reduced. If overload still occurs with 0-V bias, a "NO TEST" condition exists.

2. The current through adjacent resistors is not so large as to result in insufficient resolution for measuring the necessary current changes.

3. The equivalent resistances (as calculated by ATG) from collector-to-guard, base-to-guard, and emitter-to-guard are within the guarding capabilities of the system.

4. Proper transistor action can occur.

If all these conditions are present, a "good" test may be developed. Otherwise, no test is produced. In the ATG-developed test program, several conditions are tested. These tests will provide error messages that could appear in the test program or report file:

1. RESISTANCE FROM GUARD TO EMITTER TOO LOW

2. RESISTANCE FROM GUARD TO BASE TOO LOW

3. CURRENT SOURCE OVERLOAD

4. EMITTER CIRCUIT RESISTANCE TOO LOW

The gain test relies on the base-to-emitter voltage (VBE) of the transistor being constant for small changes in transistor current. The voltage source applies a constant bias voltage to the collector and the current source establishes a current in the base circuit. Since the voltage across the base-to-emitter and base-to-collector junctions is constant, the current through all shunt resistors (not shown in Figure 8-4) will remain constant for small changes in transistor current. Only the transistor current can change to meet the requirements of the current source. With these conditions established, the gain is measured as follows:

1. Set the current source to a nominal predetermined value (IE1).

2. Measure the current in the base circuit (IB1)

3. Slightly increase the current source to a new known value (IE2).

4. Measure the new current in the base circuit (IB2).

Calculate the gain as follows:

$$GAIN = \frac{(IE2-IE1) - (IB2-IB1)}{(IB2-IB1)}$$

where: (IE2-IE1) The change in emitter current

(IB2-IB1) The change in base current

If there are no shunt resistors across the transistor junctions, a single current reading may be made, thus speeding up the test. By means of flagspecs in the circuit description, the user may specify the operating conditions under which the transistor may be tested. Otherwise, the default values are used.

Flagspecs

Flagspec available for user control of transistor test selection are:

VBE
VCE
IC
FASTATG

VBE Base-emitter voltage for the transistor under test; default is 0.6 V.

VCE Collector-emitter voltage for the transistor under test; default is 5.0 V.

IC Collector current. Useful for specifying higher current values when testing high-power transistors. The maximum source current available is 100 mA. This will be shared by the transistor and shunt circuitry. Default is 10 mA.

FASTATG Causes ATG to execute in less time by skipping test configurations that apply a bias voltage to the collector.

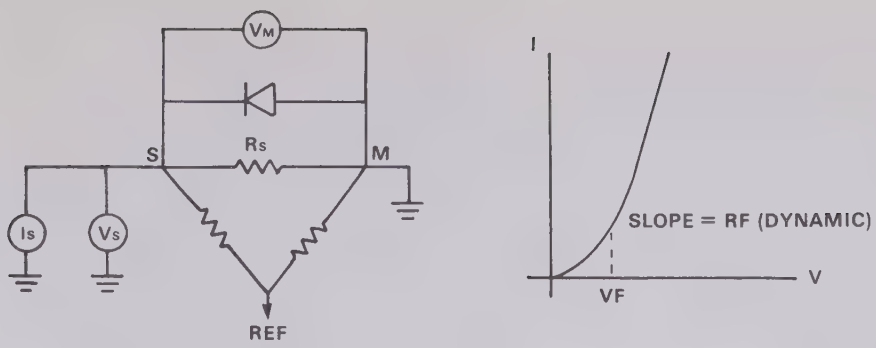


Figure 8-5. Test Configuration for Forward Knee Voltage and Dynamic Resistance Tests.

8.2.13 Diode Test

The diode testing provided by the library consists of a basic forward knee voltage test and a dynamic resistance test. These tests are evaluated by considering the application of a current limited dc source through the diode and calculating the voltage developed across it. This voltage is compared to the knee voltage value specified in the circuit description or defaults to 0.65 volt. The test current is specified by the IFV flagspec or is the default value of 20 mA. The current is then reduced to the IDR flagspec value and the voltage is again calculated. The 2 voltages and currents are used to calculate a dynamic resistance which is compared to the DR flagspec value. Figure 8-5 shows the forward knee voltage and dynamic resistance test instrumentation.

If ATG, using the IFV current, accepts the test, the test is written into the test program and is classified as a "superior" test. Otherwise, ATG will select a value of test current lower than the IFV and higher than the IDR value. Then the evaluation is repeated. If ATG, using the reduced current, accepts the test, the test is written and classified as an "adequate" test.

Any time that the forward knee voltage test is written, the dynamic resistance test is written, except when the FVONLY flagspec is specified in the circuit description. The FVONLY flagspec causes the dynamic resistance portion of the test to be omitted from the test program.

The above test philosophy is the same for both guarded and unguarded test configurations. There are 8 guarded and 4 unguarded configurations in the system library for ATG to evaluate. Their parameters and configurations are:

DC Source Voltage	Source Lead	Guarded Node	Ignored if FASTATG
HILIM+RF(IFV)+1.0 volt	Anode	Source	yes
HILIM+RF(IDR)+1.0 volt	Anode	Source	yes
-(HILIM+RF(IFV)+1.0 volt	Cathode	Source	
-(HILIM+RF(IDR)+1.0 volt	Cathode	Source	
HILIM+RF(IFV)+1.0 volt	Anode	Measure	
HILIM+RF(IDR)+1.0 volt	Anode	Measure	
-(HILIM+RF(IFV)+1.0 volt	Cathode	Measure	yes
-(HILIM+RF(IDR)+1.0 volt	Cathode	Measure	yes
HILIM+RF(IFV)+1.0 volt	Anode	N/A	
HILIM+RF(IDR)+1.0 volt	Anode	N/A	
-(HILIM+RF(IFV)+1.0 volt	Cathode	N/A	
-(HILIM+RF(IDR)+1.0 volt	Cathode	N/A	

HILIM is the voltage value represented by the knee voltage value plus the tolerance. The default value is 0.65 + 0.195 = 0.845.

RF is the dynamic resistance calculated by ATG.

IFV is the limiting current flagspec. The default value is 20 mA. ATG will set it to 40 mA if specified higher than 40 mA and will set it to 10 mA if specified less than 10 mA.

IDR is the reduced value of limiting current. The default value is 5 mA. ATG will set it to IFV -5 if IDR is specified outside the range of 5 mA to 20 mA or if it is specified less than 5 mA below IFV.

If ATG cannot find a test from any of the forward knee voltage tests, it tries to find a non-linear junction test. The non-linear junction test is evaluated by applying the current-limited voltage source, using the same instrumentation the forward voltage test uses. An unguarded test in which only direct shunt components are considered in the circuit analysis and test generated is the only test included in the system library for testing non-linear junctions. The following steps are performed to determine non-linearity.

a. The source is set high enough to turn on the diode.

b. The source is applied to the diode.

c. The voltage across the diode is measured. If the voltage is within tolerance limits, the test continues. If not, the test fails.

d. The current source is parallel with the voltage source to shunt 10 mA away from the diode.

e. The voltage across the diode is measured again. If the measured voltage is more than 50% of the source voltage, the test passes. Otherwise, the test fails, indicating that the diode is not non-linear.

If ATG selects a non-linear junction test, it is classified as an "adequate" test for the diode report statistics.

Flagspecs

Explanations of all the flagspecs that can be used when writing circuit description VALUE data for diodes are summarized below.

Flagspec	Function
DR	Nominal expected value of dynamic resistance of the diode. The default is 5.5 ohms when the forward voltage is greater than 0.5 volt; otherwise, it is 20 ohms. Used in forward voltage and dynamic resistance tests.
DRH	Maximum value of dynamic resistance. The default is 10 ohms when DR=5.5 and 30 ohms when DR=20. Otherwise, it is 1.5 times DR. If DRH is specified less than 1.2 times DR, ATG will override and use the default.

Flagspec	Function
DRL	Minimum value of dynamic resistance. The default is 0 ohm when DR=5.5 and 8 ohms when DR=20. Otherwise, it is 0.5 times DR. If DRL is specified greater than 0.8 times DR, ATG will override and use the default.
IFV	Forward voltage test current. The default is 20 mA. If it is specified less than 10 mA, ATG will override and set it to 10 mA. If it is specified greater than 40 mA, ATG will override and set it to 40 mA.
IDR	Dynamic resistance test current. The default is 5 mA. If it is specified less than 5 mA, ATG overrides and sets to 5 mA. If it is specified greater than 20 mA, ATG will override and set to 20 mA. If IFV is not greater than IDR by at least 5 mA, ATG will override and set IDR to IFV -5 mA.
FVONLY	The forward knee voltage/dynamic resistance tests are evaluated but the dynamic resistance test is not written.

The library contains a diode reverse voltage leakage test. However, it is normally ignored by ATG because the Analog Library Outline entry "G LEAKAGE CR;" is treated as a comment because of a ! that precedes the entry. The reason for this is that most circuits do not allow an accurate enough test to be useful. If the the user feels that a leakage test is useful, he can remove the ! from the user library file outline.

ATG evaluates the reverse voltage leakage tests by considering the application of a reverse voltage across the diode and calculating the current that would pass through the shunt paths around the diode. Any value actually measured greater than the shunt current would indicate a leaky diode. If ATG finds a test, it is considered "superior." Otherwise, a "format" test is written and the "unsuccessful" test statistic is recorded.

There are 4 guarded and 2 unguarded configurations in the system library for ATG to evaluate, as follows.

Source Voltage	Source Lead	Guarded Node	Ignored if FASTATG
-(VF+1.0 volt)	Anode	Source	yes
(VF+1.0 volt)	Cathode	Source	
-(VF+1.0 volt)	Anode	Measure	
(VF+1.0 volt)	Cathode	Measure	yes
-(VF+1.0 volt)	Anode	N/A	
(VF+1.0 volt)	Cathode	N/A	

VF is the specified knee voltage value which defaults to 0.65 volt.

8.2.14 Zener Diode Test

The Zener diode test is a 2-terminal, unguarded test. The dc source voltage is set to some voltage greater than the Zener voltage and the output current is limited to 20 mA. Figure 8-6 shows the test connections; Rs is the equivalent circuit resistance across the diode. Once the current limit is reached, the voltage source becomes a current source with the voltage output determined by the diode characteristics.

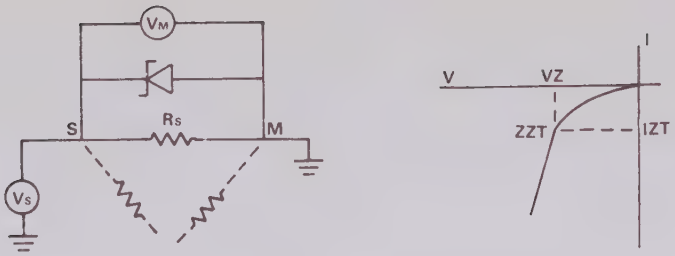


Figure 8-6. Zener Diode Test.

First, the Zener test current (IZT) and resistance (ZZT) must be determined. If these values are not specified in the circuit description, they are calculated based on the given Zener voltage (VZ) which must be specified. A 400-mW diode is assumed and the current and resistance are calculated as follows:

$$IZT\ (mA) = \frac{8.8 \times 16}{VZ\ (volts)}$$

$$ZZT\ (ohms) = \frac{17 \times VZ\ (volts)}{16}$$

If the given Zener voltage is 8 V, for example, the Zener test current would be 15.6 mA and the approximate resistance (slope of the I-V curve) at that point would be 8.5 ohms.

A source voltage (greater than the Zener voltage) with a 20-mA current limit is then assigned. ATG evaluates the resulting test configuration as follows:

1. If the required node voltage is greater than 16 V (the maximum output of the voltage source), then a "NO TEST" condition exists.

2. If the available diode current is less than 2 mA due to shunt circuit circuit paths across the source, then the source and measure terminals are interchanged and the source polarity reversed. If this new configuration still does not provide more than 2 mA of diode current, again a "NO TEST" condition exists because of the circuit environment.

3. If the voltage and current values are acceptable, then ATG determines whether the actual diode current is less than or greater than IZT and sets error limits accordingly. These pass limits take into consideration the Zener voltage, the component tolerance and the voltage drop caused by the actual test current through the diode. If not otherwise specified, tolerance is assumed to be 10%.

8.2.15 SCR Test

The ATG test library contains two tests for the SCR based on the instrument configurations shown in Figure 8-7. One of these tests (SCRIM) is used for the low current limit of the voltage source and produces instrument configuration (A). The other test (SCRVM) is used for the high current limit of the voltage source and produces instrument configuration (B). In both cases, all equivalent resistances at each node are considered in the test evaluation.

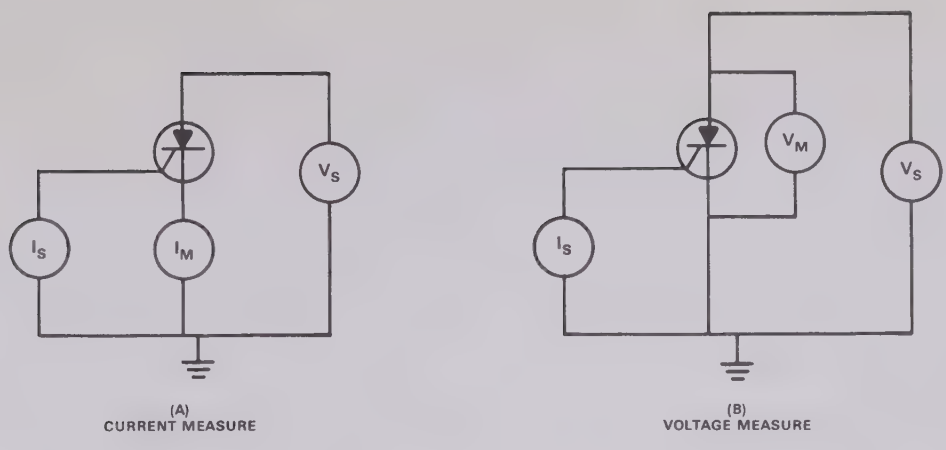


Figure 8-7. SCR Test Configurations.

In evaluating the various test configurations, ATG checks to insure that:

1. The total current the voltage source must supply does not exceed the current limit set for the voltage source.

2. The current source does not have to supply more than 10 percent of its specified value to shunt current paths. If it does, the gate current will be increased by ATG to compensate for the shunt paths.

3. The gate current does not exceed 40 mA.

If all these conditions are met, a test can be developed.

The instrument configuration to be used is determined by the voltage source current limit (ILIM) specified in the circuit description. If none is specified, the test defaults to the low limit (20 mA) and the configuration in Figure 8-7 (A) is used. This configuration verifies proper SCR operation by measuring the flow of current in the cathode. The current measurements are made as follows:

1. Anode voltage is applied with the voltage source. Gate current is set to zero.

2. The "off" cathode current is measured.

3. A current source applies gate trigger current to start SCR conduction.

4. The "on" cathode current is measured.

5. The gate trigger current is removed.

6. The subsequent holding current is measured to verify that the SCR is still conducting.

If the high current limit for the voltage source is specified in the circuit description, the instrument configuration in Figure 8-7 (B) will be used. This configuration verifies proper SCR operation by measuring the anode-to-cathode voltage drop. These measurements are made as follows:

1. Anode voltage is applied with the voltage source. Gate current is set to zero.

2. The "off" anode-to-cathode voltage is measured.

3. A current source applies gate trigger current to start SCR conduction.

4. The "on" voltage is measured.

5. The gate current is removed.

6. The holding voltage is measured to verify that the SCR is still conducting.

SCR Flagspecs

Flagspecs available for user control of SCR test selection are:

VA Anode voltage for the SCR under test. Default is 4.0 volts. The limits are 0 to 16 volts.

ILIM Current limit for the voltage source. The only values that can be selected are 0.02 A and 0.128 A. Default is 0.02 A.

IGT Gate current used to trigger the SCR. Default is 2 mA. If a value less than 0.1 mA or greater than 10 mA is specified, ATG will override and use 2 mA.

8.2.16 Operational Amplifier Test

The operational amplifier test uses the instrument configuration shown in Figure 8-8. The ATG Test Library contains only one test for operational amplifiers, written for the LM301. However, this test can be modified to provide library procedures for other amplifiers, and comments are provided in the file to guide the user in making the necessary alterations.

Operational amplifiers are tested with power applied to the circuit board. Small voltage differentials are applied to the amplifier's inverting and non-inverting inputs from low impedance sources (to override feedback elements), and the resulting output voltage is measured. To pass the test, the amplifier's output voltage must rise higher than a specified value and fall lower than a second value. The user supplies two flagspec values, POSRAIL and NEGRAIL, in the .CKT file. Their values will usually be the positive and negative supply voltages. With a single positive power supply, NEGRAIL is specified as zero; for a negative supply, POSRAIL is zero. The actual test limits are calculated as flagspec $\pm$ 10 %, which will provide a good test in most circuits. In some cases, however, such as clamping circuits, the maximum and minimum output voltages may be limited by factors other than the operational amplifier's internal saturation voltages. In such cases, values of POSRAIL and NEGRAIL must be chosen accordingly (10 % higher or lower than the expected maximum swing). Likewise, if supply voltages are changed, POSRAIL and NEGRAIL must adjusted as well.

The LM301 test can be used for all package types of the LM301.

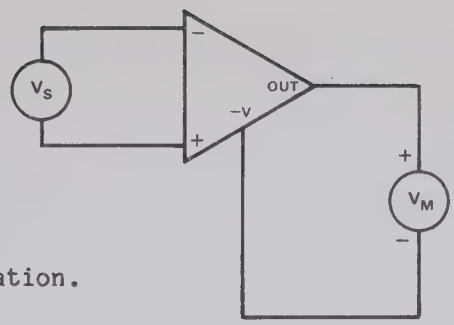


Figure 8-8.
Operational Amplifier Test Configuration.

In evaluating the test configuration it chooses, ATG checks to ensure that:

1. The feedback impedance from the output of the opamp to the inverting input is greater than 1000 ohms.

2. The current in the output circuit is less than 200 mA.

If these conditions are met, a test can be developed for the opamp. If either of these conditions is not met, a non-executing FORM test will be written in the test program and a message will appear in the report file indicating which condition failed.

The opamp test verifies proper operation of the component by applying differential input voltages to force the output to the supply voltages. The exact procedure is as follows:

1. The voltage source applies a small positive voltage (VDIFF flagspec) between the non-inverting and inverting inputs of the opamp.

2. The output of the opamp will now be near the negative supply voltage (the NEGRAIL flagspec). The output voltage is measured with the voltmeter connected between the output and the negative supply of the opamp but is calculated with respect to ground.

3. The voltage source now applies a small negative voltage differential to the inputs (-VDIFF).

4. The output of the opamp should now be near the positive supply voltage (the POSRAIL flagspec). The output voltage is measured again. The voltmeter should find that the opamp has swung to its maximum output voltage.

If the opamp is operating from a single power supply, only one voltage measurement will be made. With a positive supply, the positive swing is tested; with a negative supply, the negative swing is tested.

Example

Because the opamp is tested with power applied, the opamp power supplies must be turned on before the test is executed. It is recommended that the .ATO or .ATX file be used to insert the power supply statements into the test program. The INCLUDE AFTER function can be used. The stream number for the INCLUDE is 100. The following is an example of this .ATO or .ATX implementation.

The supply voltages for the opamp are -15 volts and +15 volts. These voltages are to be supplied by UPS(4) and TPS(2) level B. The .ATO or .ATX file entry would be:

```
INCLUDE AFTER (100)[
SET PIO(0) HRLY(CLOSE 3); /*GROUND RELAY*/
SET PIO(0) HRLY(CLOSE 7); /*TPS ISOLATION RELAY*/
SET PIO(0) HRLY(CLOSE 8); /*UPS ISOLATION RELAY*/
SET DCV UPS(4) V=-15 I=1 DLY=50M
  [WRITE 'UPS(4) DID NOT SETTLE';
   BRANCH CLRPOWER;];
SET TPS(2) VB DLY=50M;
]
```

Meaning: The PIO statements close the ground and isolation relays. UPS(4) is programmed to -15 volts with a current limit of 1 amp. If UPS(4) fails to settle, the program will branch to the label CLRPOWER. TPS(2) is activated to output voltage level B. These statements will now appear in the test program at the label SETPOWER in addition to the comments the test library generates. A complete discussion of .ATO and .ATX file usage can be found in the System Operation manual. The Orientation Notes manual contains additional power supply programming examples.

The Test Library will provide the following statements at the CLRPOWER label in the test program.

```
CLRPOWER:CLEAR UPS;
         CLEAR TPS;
         SET PIO(0) HRLY(OPEN 3); /*GROUND RELAY*/
         SET PIO(0) HRLY(OPEN 7,8); /*ISOLATION RELAYS*/
```

If any other statements are required here, the .ATO or .ATX file can be implemented again. The stream number is 110. All power supplies should be cleared before the scanners and multiplexer are cleared (label ADONE) to prevent damage to the relays.

Flagspecs

Flagspecs available for user control of opamp test selection are:

POSRAIL Normally the positive supply voltage. If only a negative supply is used, set to zero. Results in a minimum (lowest) acceptable positive voltage, referred to ground, that the output must reach when the amplifier is driven positive. The actual test limit is POSRAIL - 10 %. This flagspec must be specified. No default is provided.

NEGRAIL Normally the negative supply voltage. If only a positive supply is used, set to zero. Results in a maximum (highest) acceptable negative voltage, referred to ground, that the output must reach when the amplifier is driven negative. The actual test limit is NEGRAIL + 10 %. This flagspec must be specified. No default is provided.

VDIFF Differential input voltage. This is an optional flagspec. ATG provides a default value of 100 mV.

The values for POSRAIL and NEGRAIL do not have to be near the supply voltages. If the opamp has circuitry on its output to prevent it from going to the supply voltages (e.g., Zener diodes), the maximum output voltages should be specified in the flagspecs.

Testing Other Types of Opamps

The test in the system Test Library is only for the LM301 opamp. However, it is possible the same technique will work for other opamp types. The LM301 test can be edited to produce a test for a different opamp. The process for this editing is discussed below.

Assume the LM301 test in the system library is to be enhanced to include testing of an LM741 in an 8-pin DIP package.

1. The test procedure must be extracted from the system test library source file (TSTLIB721.LBS). The test name is RAIL.

2. Near the beginning of the procedure a series of checks (IF MYTYPE) is done to determine which package type is to be tested. An additional check for the LM741 must be added. The following comment appears in the procedure:

```
!CHECKS FOR OTHER TYPES OF
!OPAMPS SHOULD BE ADDED HERE
```

Immediately following this comment and before the .QUIT IGNORE statement, the following statements should be added:

```
.IF MYTYPE(LM741);      !8 PIN DIP
        .PIN 1=2;       !INVERTING INPUT
        .PIN 2=3;       !NONINVERTING INPUT
        .PIN 3=6;       !OUTPUT
        .PIN 4=4;       !MINUS SUPPLY VOLTAGE
.ELSE;
```

3. After the .QUIT IGNORE statement, add the following statement:

```
.FI;
```

4. Repeat this process for the form test, RAILFOR.

5. The test library OUTLINE must be modified to include the following entry:

```
G OPAMP LM741;
```

In this case, no changes are necessary to any directories.

6. These are the only changes necessary in the test library. The two new tests, RAIL and RAILFOR, and the new OUTLINE are now inserted into a user test library.

7. An entry must be made in a user component library for the LM741. This will allow the CDL process to recognize LM741 in the circuit description. The source file (.ACS) for the LM741 would be:

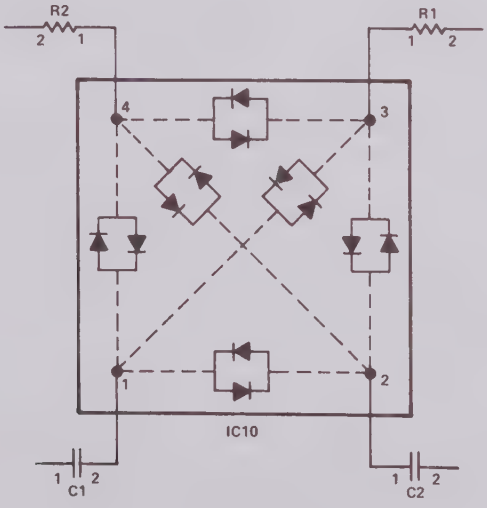
```
%CIRCUIT;
1 EXT ,N1 U1.1;
2 EXT ,N2 U1.2;
3 EXT ,N3 U1.3;
4 EXT ,N4 U1.4;
5 EXT ,N5 U1.5;
6 EXT ,N6 U1.6;
7 EXT ,N7 U1.7;
8 EXT ,N8 U1.8;
U1 UK8 1=N1,2=N2,3=N3,4=N4,5=N5,6=N6,7=N7,8=N8;
```

8. The LM741 component source is inserted into a user component library (.ACL).

9. An LM741 can now be entered in the circuit description.

8.2.17 Integrated Circuits

There are three categories of ICs: (1) digital circuits which have a test in the Digital ATG Test Library; (2) digital circuits that do not have a test in the Digital ATG Test Library; and (3) analog circuits. In the first case, ATG will process the Digital Library routine specified by the IC type and write the resulting test procedure into the test program. In the second case, ATG will create data for the digital diagnostic data base file (.IDD or .IDX) even though there is no test written. This allows for use of the Scratchprobing technique at run-time. In the third case, ATG considers the IC only as it would affect analog tests of surrounding components. The respective CDL symbols for the circuit description file are (1) [type of IC], (2) DICnn, and (3) ICnn.



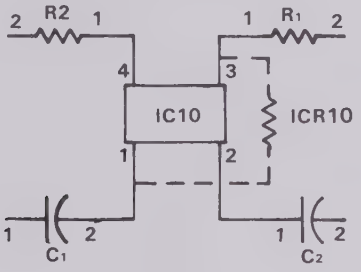

```
% Circuit;

IC10   IC   C1.2, C2.1, R1.1, R2.1;
```

Figure 8-9. Back-to-Back Diode Model of IC in Analog Test.

In developing analog tests, ATG will model all ICs as a circuit with back-to-back silicon diodes between all possible pairs of device pins. Figure 8-10 illustrates this model. Only pins listed in the CDL or defined in the ATG test library can be modeled this way. This modeling technique can only be used where the device pin has a test head nail connected to it that has been specified as analog or as hybrid in the CDL. Because of this model, ATG only uses low voltage (0.2-V) tests on components connected to an IC pin. This prevents diode junctions in the IC from affecting the measurement. An example of a circuit description for an IC that is not included in the Digital ATG Test Library is shown in Figure 8-9.

The user may check IC orientation by testing either a known resistance or a diode characteristic between 2 IC pins.

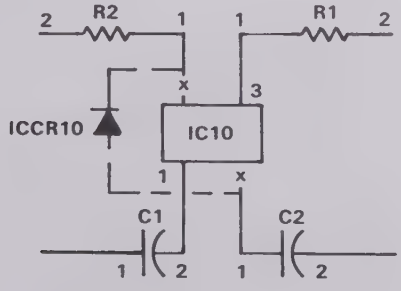
To do an orientation test based on an internal resistance (R) between 2 pins, add to the circuit description a "dummy" resistor of "R" ohms between the two pins, as shown in Figure 8-10. The system will measure this resistance. A passing test will indicate correct insertion of the IC.




```
IC10    IC4   C1.2, C2.1, R1.1, R2.1;
ICR10   R     IC10.3, IC10.1;
```

Figure 8-10. Example of IC Orientation Test (with Known Resistance).

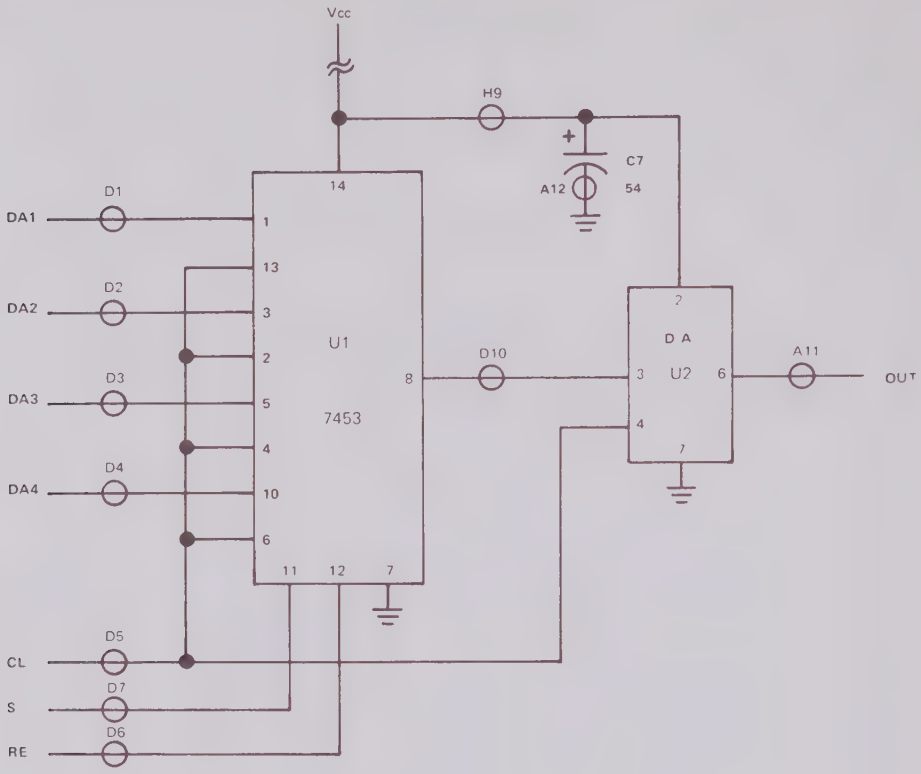
To do an orientation test based on a known diode characteristic between 2 pins, the 2 pins must be left out of the circuit description. This prevents the modeled back-to-back diodes between these two pins from affecting the ATG test evaluation. Then, an external "dummy" diode must be added to the circuit description, as shown in Figure 8-11. The system will perform a diode test to determine if the IC is properly inserted.




```
IC10    IC4   C1.2, 3 = R1.1;
ICCR10  CR    A = C2.1, C = R2.1;
R2      R     ICCR10.C, other connections;
C2      C     ICCR10.A, other connections;
```

Figure 8-11. Example of IC Orientation Test (with Known Diode Characteristic).

Figure 8-12 illustrates the CDL for a circuit with 2 ICs. One is a digital IC which is included in the Digital ATG Test Library. The other is a d/a circuit that must be considered for analog testing.




```
% CIRCUIT;

   DA1      EXT     ,U1.1,
   DA2      EXT     ,U1.3,
   DA3      EXT     ,U1.5;
   DA4      EXT     ,U1.10;
   S        EXT     ,U1.11;
   RE       EXT     ,U1.12;
   VCC      EXT     ,U1.14  U2.2  C7.P;
   GND      EXT     ,U1.7  U2.7  C7.N;
   CL       EXT     ,U1.13  U1.2  U1.4  U1.6  U2.4,
   OUT      EXT     ,U2.6;
   U1       7453    DA1, CL, DA2, CL, DA3, CL, GND, U2.3, 10 = DA4, S, RE, 14 = VCC;
   U2       IC8     2 = VCC, U1.8, CL, 6 = OUT, GND;
   C7       CP      P = VCC, N = GND,

% VALUE. 2270;

   C7       = 54U;

% ADAPTOR: 2270,

   D1       = U1 1,
   D2       = U1 3,
   D3       = U1 5;
   D4       = U1 10,
   D5       = CL,
   D6       = U1 12,
   D7       = U1 11,
   H9       = VCC,
   D10      = U1 8,
   A11      = OUT,
   A12      = GND,
```

Figure 8-12. Sample IC Circuit Description.

Microprocessor Tests – Appendix A

A.1 INTRODUCTION

Testing most ICs on the 227X involves little more than allowing ATG to search for the proper test in the digital library. However, some of the microprocessors convered by standard library tests require additional clocks or specially conditioned signals. This is accomplished by adding combinations of peripheral chips and buffers to the test fixture. Users with the SpeedPlus testing option will ususally not need to use additional fixture hardware unless they choose not to test a device with the SpeedPlus hardware.

This appendix gives specific recommendations for testing 5 popular microprocessor families: Intel's 8080/8085, 8048, and 8086 series, MOS Technology's 6500 series and Texas Instruments' TMS9900 series. Each paragraph is devoted to one family of processors and includes a wiring diagram for the additional test fixture hardware as well as a brief explanation of how the circuit works.

The paragraph on the Intel devices describes a method of single-stepping the microprocessors, using a flip-flop in the test fixture. For the 8080, an 8224 clock chip must also be provided in the fixture, unless a clock chip is available on the board-under-test. The 8085 needs only a flip-flop in the test fixture, since it has its own clock. The MCS 48 series processors also require just a flip-flop that is used to drive the processor's single-step (SS) input.

The MOS Technology chips need a buffer to insure the integrity of the clock pulse. The 6512 and 6513-15 require 2-phase clock inputs, but a buffer with complementary outputs can supply both lines. The Texas Instruments device may require a TMS9904 clock driver.

When testing boards with multiple clocks, be sure that the test program disables as many clocks as possible to avoid noise problems. This can be done using the Automatic Test Options (.ATO) feature, as described in the Test Set Preparation manual, Section 5.

A.2 TESTING INTEL'S MCS 80/85 PROCESSOR FAMILY

A.2.1 8080

To meet the 8080's clock requirements, an Intel 8224 chip can be used with a single-step circuit to supply the clock signal. A simple circuit using a D-type flip-flop performs the single-step. The procedure for constructing the test fixture is:

a. Wire the circuit as shown in Figure A-1. This circuit diagram shows all the component wiring. The physical location of the components is left to the fixture designer, but use short, twisted-pair clock lines to ensure a good, clean signal. Also, if an 8224 is used in the fixture, verify that any other on-board clocks or oscillators are disabled by the test program.

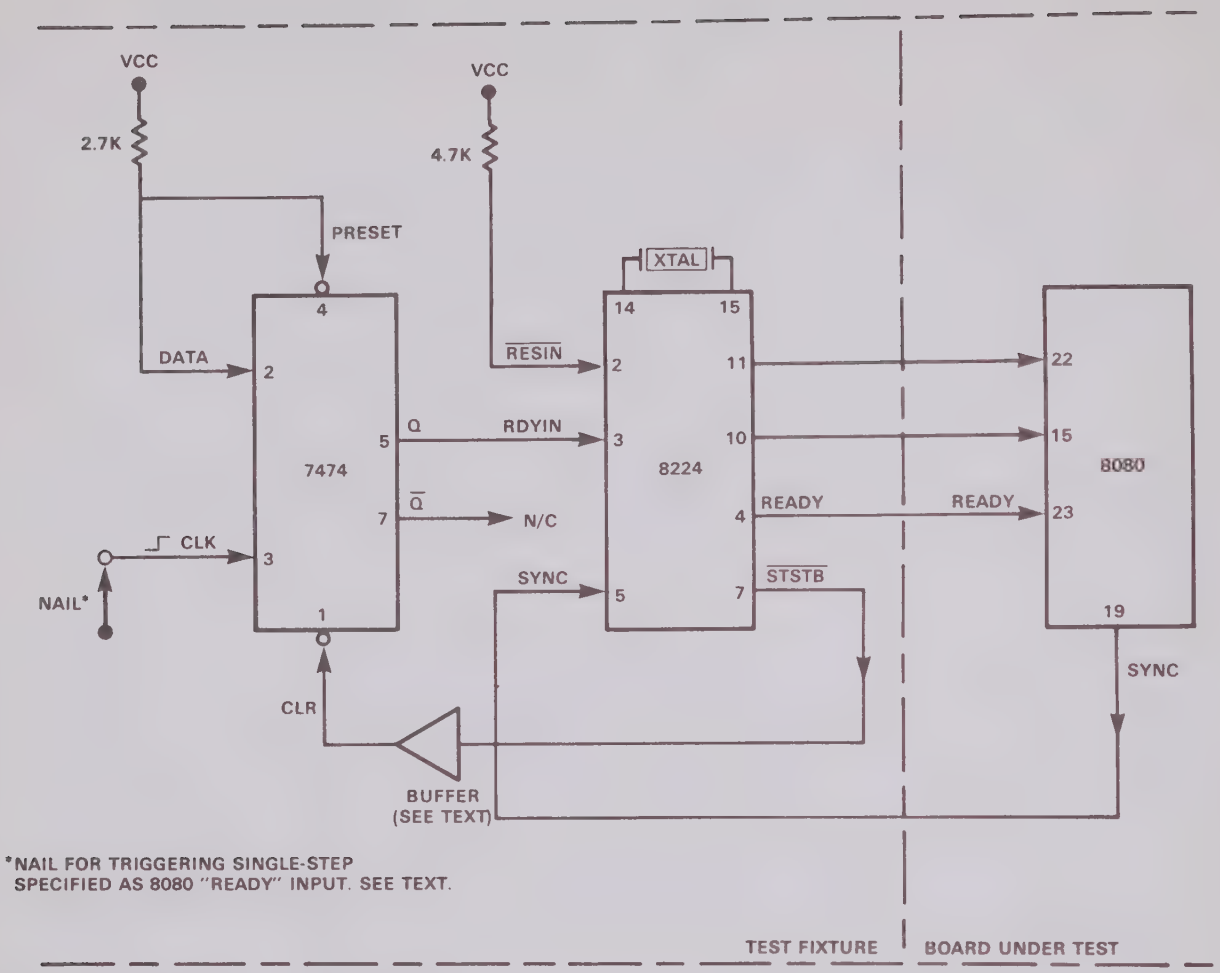


Figure A-1. The 8080 Single-Step Circuit.

b. Specify nails in the circuit description file as for any other chip, except for the nail on the CLK input (pin 3) of the 7474 flip-flop. This nail is specified in the circuit description as if it were on the READY pin of the 8080.

c. Use the ATO feature to disable any unnecessary clock signals during testing. The fixture-mounted clock should also be disable when components other than the 8080 are being tested. If this is not done, clock noise could cause intermittent failures. The Automatic Test Options programming feature is described in the Test Set Preparation manual, Section 5.

If the board-under-test has an 8224 clock chip on it, one in the fixture may be unnecessary. Just follow the wiring instructions above, but alter the test program to used the on-board clock instead of the fixture-mounted clock in shown Figure A-1.

The key to testing 8080s on the 227X Board Test Workstation is the ability to shut off the microprocessor after a single clock cycle. A D/S nail placed on the flip-flop's clock input is tied to the 8080's READY input via the flip-flop's Q output and the 8224's RDYIN and READY output. The shut-off timing

is provided by the SYNC output of the 8080. This pin is tied to the flip-flop's CLR input through the 8224's STSTB (to invert the pulse). When SYNC triggers, STSTB clears the flip-flop, resetting Q and disabling the 8080 through RDYIN and READY. Each cycle of the flip-flop causes a the microprocessor to execute a single read or write cycle.

A.2.2 8085

The 8085 incorporates its own high-speed clock (essentially an on-chip 8224), which makes test synchronization difficult.

The test fixture circuit synchronizes the tester to the 8085 by enabling the 8085, 1 memory cycle at a time. The driver/sensor nail on the flip-flop's clock input is tied (through its Q output) to the 8085's READY input. The ALE signal from the 8085 is fed back to the flip-flop's CLR input at the end of each memory cycle, resetting the Q output and disabling the microprocessor through READY.

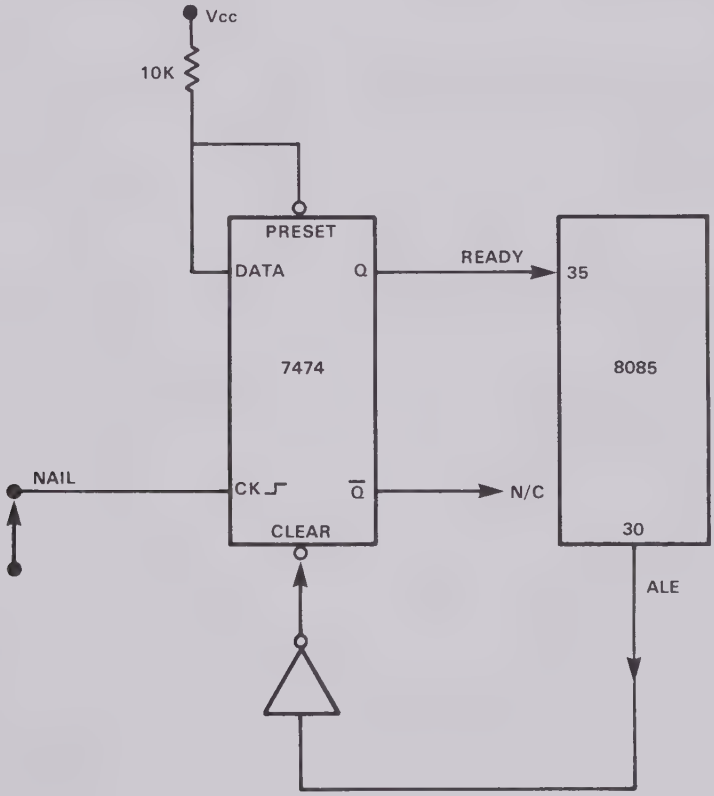


Figure A-2. The 8085 Single-Step Circuit.

The procedure is:

a. Provide the 8085 with a free-running oscillator, either using the one on the board-under-test or mounting one in the test fixture. If the oscillator is mounted in the fixture, remember to keep the lines as short as possible to ensure a clean signal.

b. Build the circuit shown in Figure A-2. This circuit synchronizes the tester to the processor by enabling the clock 1 memory cycle at a time. The

Address Latch Enable (ALE) signal from the 8085 is fed back to the flip-flop's CLR input at the end of each memory cycle, disabling the clock.

c. Type in the circuit description as usual, except for the nail on the flip-flop's CLK input. This nail is entered as if it were on the READY input of the 8085.

A.3 TESTING INTEL'S 8048 MICROPROCESSOR FAMILY

The 8048 series microprocessors use on-board clocks, which makes it difficult to synchronize them with the 227X. However, these chips include a single-step input (SS) to ease testing.

When the tester enables the flip-flop, shown in Figure A-3, SS is driven high, starting the processor. When the ALE goes low, the flip-flop clears, bringing the SS input of the processor low and stopping the microprocessor. Using SS and ALE, a single-step circuit is easily constructed with a D-type flip-flop and is used as follows:

a. Build the circuit diagrammed in Figure A-3. The flip-flop is wired with the PRESET tied high and the CLR line driven by ALE. ALE is buffered because the CLR input of an SN7474 represents 3 TTL loads. If a different flip-flop is used, the buffer may not be necessary. XTAL1 AND XTAL2 of the 8048 must be provided with a crystal, or be driven by a free-running oscillator. Refer to the 8085 test listing in the digital test library.

b. Specify the fixture nails as usual in the .CKT file, except for the flip-flop's CLK input, which is specified as the microprocessor's SS input.

c. Use ATO to disable any unnecessary clocks during the test. In testing memories or other components, extra clock signals on the board can cause intermittent faults. During test bursts, fixture nails should be disconnected from crystals to ensure clean signals.

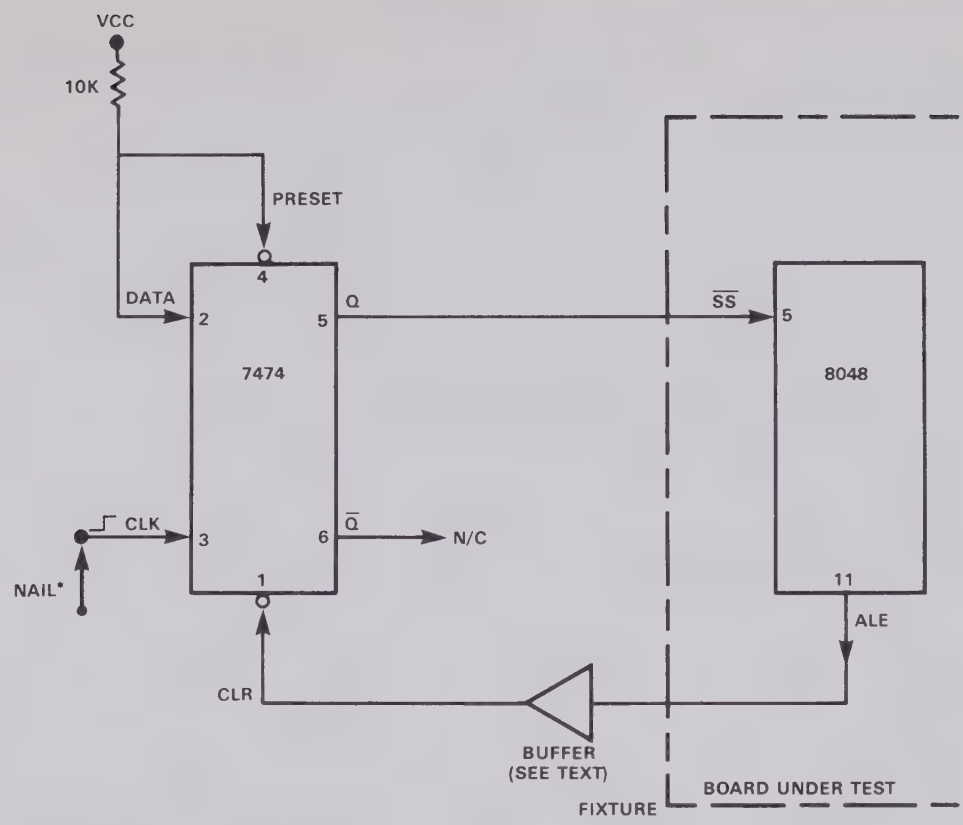


Figure A-3. The 8048 Family Single-Step Circuit.

A.4 TESTING INTEL'S 8086 FAMILY OF MICROPROCESSORS

The 8086 is a 16-bit Central Processing Unit with a 16-bit data path connecting memory, I/O and the processor. It can address up to a megabyte of memory, divided into 64-kbyte segments.

The 8088 is identical to the 8086 in its internal organization, its instruction set and its control signals. The only difference is an 8-bit data path.

The 8089 Input/Output Processor is a 16-bit I/O system on a single chip. The 8089 consists of two independent I/O channels, each of which is capable of functioning as a CPU and DMA Controller. When in MAX mode, the 8089 is compatible with both the 8086 and 8088.

Devices in the 8086 microprocessor family require extra fixture circuitry for testing with standard digital library tests. This extra circuitry is mounted in the test fixture and 'single-steps' the device under test.

Both the 8086 and the 8088 provide a 'ready' input which serves to place the processor in a wait state. By properly generating and synchronizing a ready pulse, these devices can be stepped through a test procedure. Figure A-4 illustrates the need for a single step circuit. For the 8086 or 8088 running at 5 MHz, the required input pulse width on the ready pin should be less than 600 ns to guarantee that the device performs only one bus cycle. However, the minimum pulse obtainable from a 227X driver is 3.5 us.

Figure A-5 shows how this ready pulse must be synchronized to the clock which is driving the 8086/88. A single-step circuit was developed to meet these requirements. This circuit uses a 74121 monostable multivibrator connected as a one-shot, as shown in Figure A-5. When the input to the 74121 makes a low-to-high transition, a high pulse is generated at the output. This is the ready signal to the 8086/88. Since the pulse width depends on what frequency the device is being run at, provisions were made for adjustment with the addition of this pot. By trimming the pot, the ready pulse can be adjusted to meet the requirements determined by the CPU operating frequency.

Using the ready input of an 8284 solves any problem of pulse synchronization. The ready input of the 8284 insures that the ready pulse reaches the input of the 8086 when the CPU clock is stable.

If there is no on-board 8284, or if either of the ready inputs is not available, the ready pulse can be synchronized using a 7474 D flip-flop, as shown in Figure 2-B. Connecting the clock input of the F/F to the CPU clock insures that the ready pulse is properly synchronized.

When the 74121 is used in conjunction with an on-board clock, one extra fixture IC is needed in the fixture hardware (Figure A-6). This setup does not rely on the ALE signal of the 8086, so it will test both max and min modes with no modification. The use of bus transceivers on the multiplexed buses of these devices is optional.

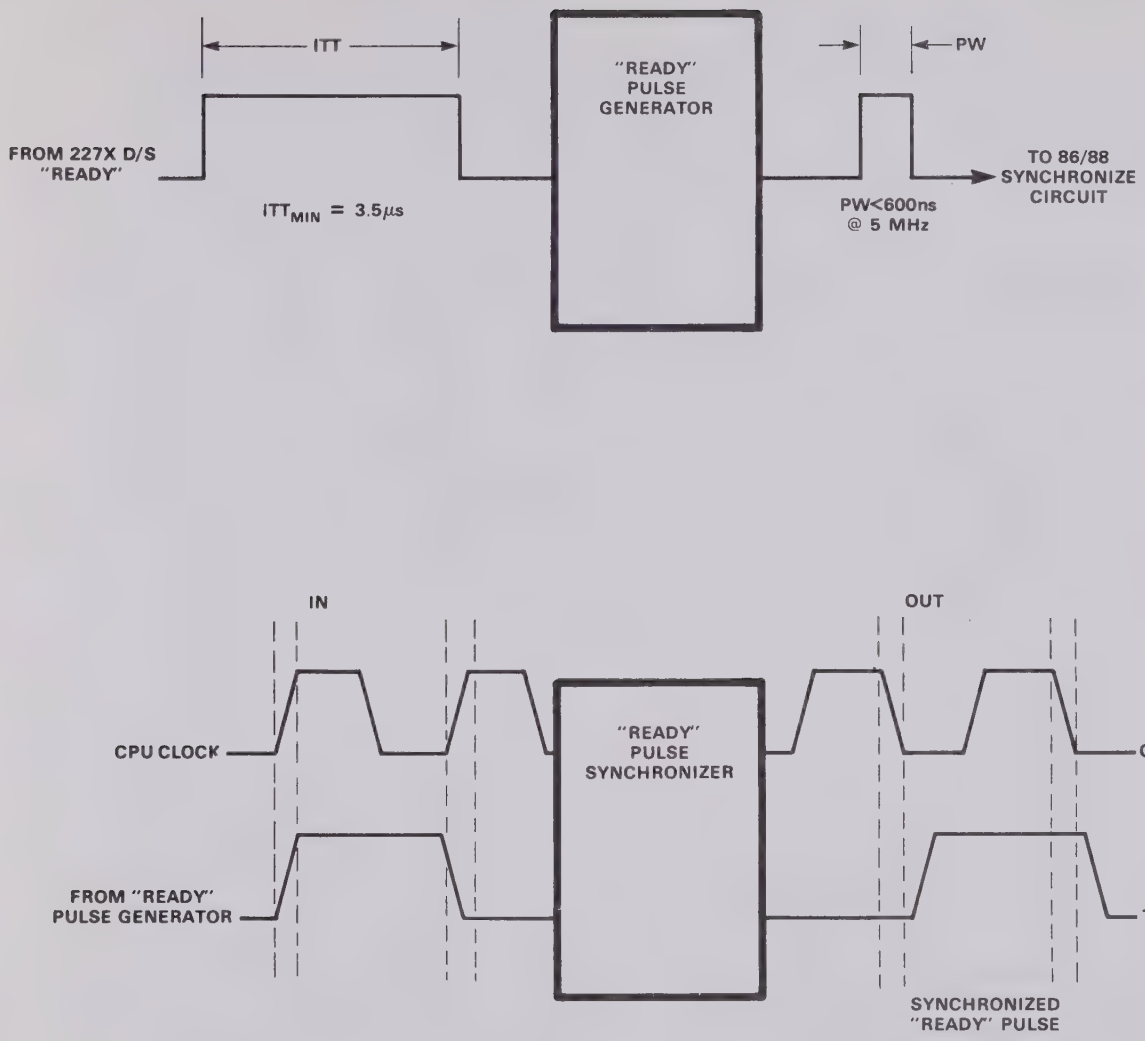


Figure A-4. Single-Step Requirements.

The procedure for testing the 8086 and 8088, when configured in MIN mode, is:

a. Wire the circuit as shown in Figure A-4, keeping clock lines as short as possible. If an 8284 clock generator is used in the fixture, make sure any on-board clock generators can be disabled.

b. Specify all nails in the circuit description file as usual, except for the nail on the clock input of the 74121. This nail is specified as if it were on the ready input of the 8086/8088. Also, if a nail is used on the clock line to test a series resistor or for shorts testing, include a relay on the nail and make sure the relay is open when testing the 8086/8088.

c. If the 8284 is used in the fixture, use the ATO feature to disable any on-board clock signals when testing the 8086/8088.

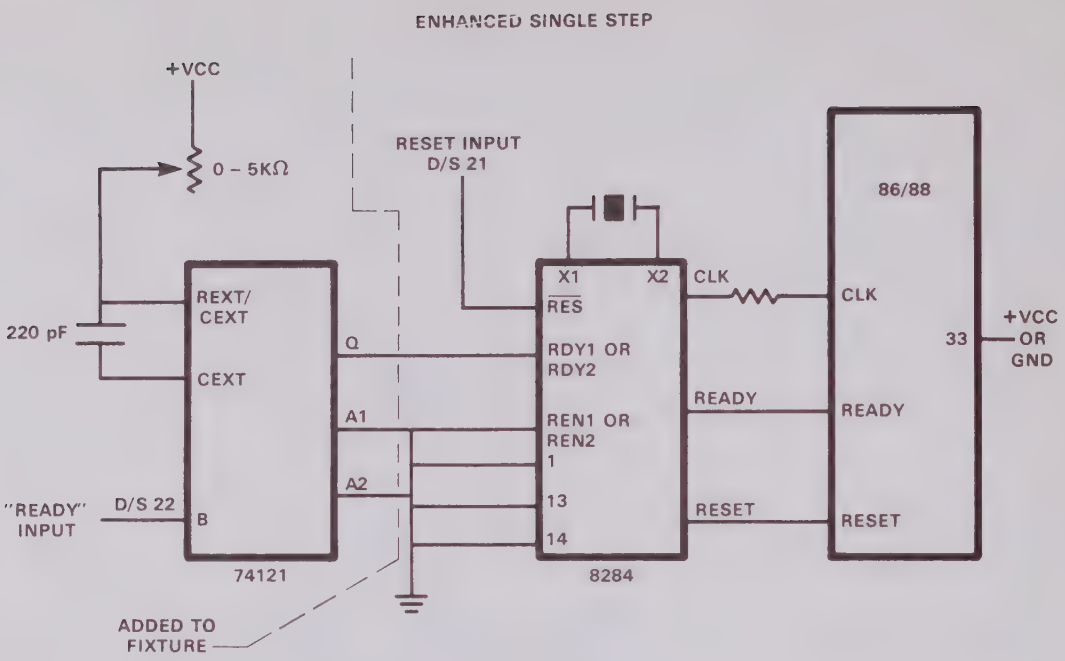


Figure A-5. Enhanced Single-Step Circuit.

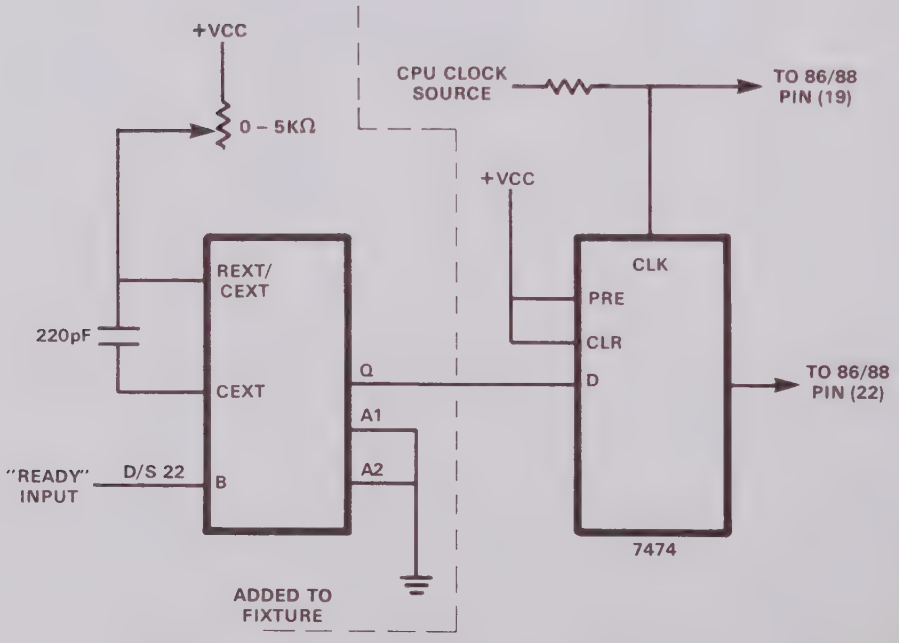


Figure A-6. Synchronizing Circuit without 8284.

A.5 TESTING MOS TECHNOLOGY'S 6500 FAMILY

The MOS 6500 series microprocessors require clocks with very fast rise and fall times. To ensure clean, fast clock signals the clock inputs to these chips should be buffered.

Figure A-7 shows how to wire the buffers for chips 6502 through 6506 of the series. All use single-phase clock inputs.

Figure A-8 shows how to wire MOS 6513 through 6515, all of which require 2 clock phases. The only difference between Figures A-7 and A-8 is that, in Figure A-8, the buffer has complementary outputs.

The MOS 6512 requires a buffer with complementary outputs like the 6513 through 6515 series. In addition to this, the DBE input must also be driven by the buffer, as shown in Figure A-9. DBE is another phase 2 clock input.

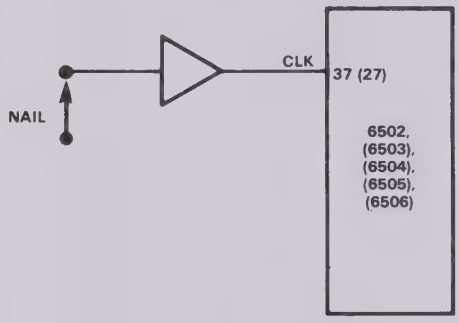


Figure A-7. Single-Step Circuit for MOS Technology Chips 6502 through 6506.

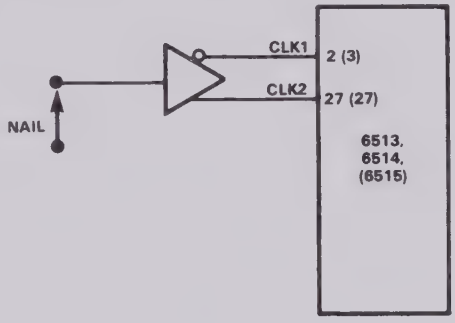


Figure A-8. Single-Step Circuit for MOS Technology Chips 6513 through 6516.

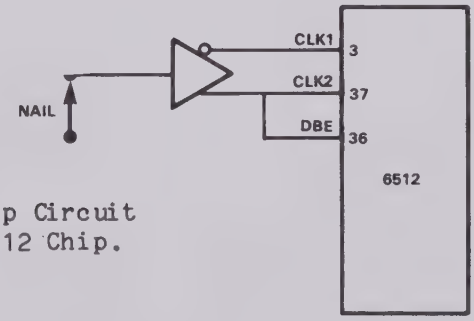


Figure A-9. Single-Step Circuit for MOS Technology's 6512 Chip.

A.6 TESTING TEXAS INSTRUMENTS' TMS9900

The TMS9900 microprocessor requires four-phase clocking at 2-3 MHz. To test a microprocessor with these electrical requirements, extra circuitry is necessary (Figure A-10) to single step the device. This circuitry must be included in the fixture. If the FFD input and FFQ output of the on-board TMS9904 clock generator are available for use as diagrammed, then another TMS9904 is not needed in the fixture. The procedure is:

a. Wire the circuit as in Figure A-10, keeping clock lines as short as possible. If a TMS9904 clock generator is used in the fixture, make sure that the on-board clock generator can be disabled. If the on-board TMS9900 clock generator is used, wire the FFD input and FFQ output as in Figure A-10.

b. Specify all nails in the circuit description file as usual, except for the nail on the clock input of the 7474. This nail is specified as if it were on the READY input of the TMS9900. Also, if nails are used on the clock lines to test any series resistors or for shorts testing, include relays with the nails and make sure the relays are open when testing the TMS9900.

c. If the TMS9904 clock generator is used in the fixture, use the ATO feature to disable any on-board clock signals when testing the TMS9900.

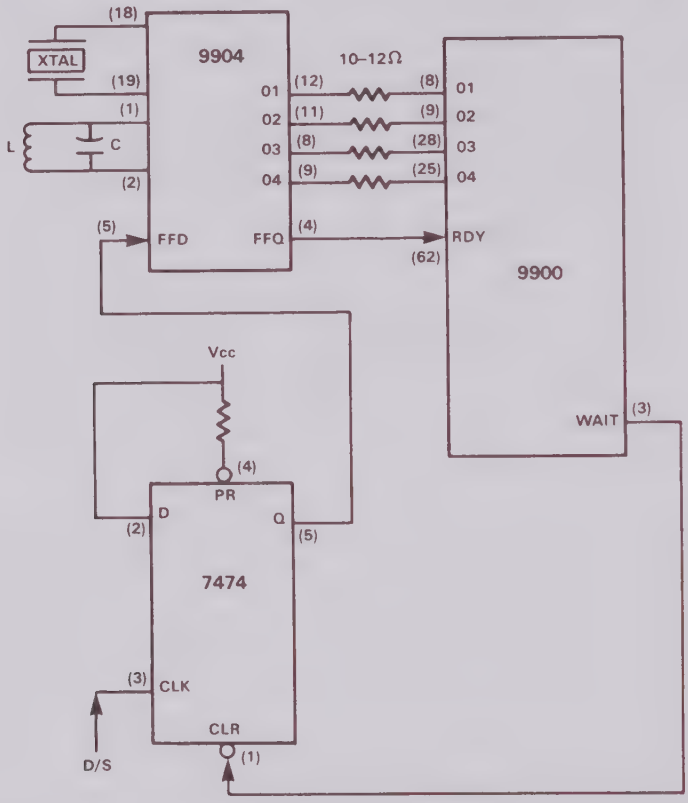


Figure A-10. The Single-Step Circuit for the TMS9900.

This test for the 9900 was developed using a 48-MHz, third overtone crystal for 3-MHz operation. The values of the capacitor and the inductor in the tank circuit were computed using the following formula:

where:

When testing the TMS9900, refer to TI's 9900 Family Systems Design and Data Book.

Due to the sensitivity of the TMS9900, the input high-voltage threshold (VIHB) should be lowered to avoid damaging the TMS9900 during testing. Logic family B can be used for this purpose. VIHB should be lowered from the default of 4.7 V to between 3.0 and 3.5 V. The remaining voltage thresholds (VILB,

VOHB, VOLB) can remain at their default values.

The TMS9900 requires 3 power-supply voltages, -5 V, +5 V and +12 V. To prevent damaging the TMS9900 during power up, the negative supply (-5 V) should be turned on first, then the +5V supply, and, finally, the +12 V supply. Also, include in the test program a 20-ms delay after each supply is turned on.

The ATO feature can be used for the power-up sequence and the assignment of logic family B, as outlined above.

Index

--A--

ACL, 1-1
ACS, 7-2
ADAPTOR section, 8-8
ALL, 4-3
ASSIGN, 5-27
ATL, 1-1
ATL, management options, 4-2
Alias, 4-9
Analog ATG processing, 2-6
Analog Auto Test Generator, 2-1
Analog Test Selection Language (ATSL), 5-1, 25
Analog component librarian (ACL), 3-4, 4-8
Analog component library, 7-1
Analog library processing, 5-1, 9
Analog test evaluation, 5-12
Analog test librarian (ATL), 4-1

--B--

Bus tests, 6-1, 17

--C--

CDL mode, 7-2
CERR, 8-4, 15
CHANGE mode, 4-2
CHIRESID, 8-15
CHOOSE Call, 6-58
CHOOSE Insert, 6-59
CLOCK, 6-13, 60
CLORESID, 8-15
CLUSTer, 6-62
CONFIGURATION, 5-6, 15
CONNected, 6-64
CONTEXT, 6-64
CREATE mode, 3-3, 4-2
Capacitance test, 8-13
Circuit Description Compiler, 2-1
Circuit Description Extractor (CDE), 2-1
Circuit condition, 8-5
Clock requirements, 6-40
Complex analog components, 7-1
Component description, 7-1
Component description file (ACS), 7-2
Component name, 4-8
Connectivity test procedures, 8-8
Connectivity tests, 2-6
Context pin, 6-7

--D--

DCR, 8-11
DAI Digital ATG, 2-4
DECLARE GROUP, 6-14
DECLARE, 6-13
DEFINE, 5-21, 28
DEFined, 6-65
DEV, 4-1
DH (default high), 6-66
DIRECTORY, 4-3, 5-3, 13
DISABle, 6-14, 67
DISJoint, 6-68
DL (default low), 6-68
DR, 8-20
DRH, 8-20
DRL, 8-21
DTG transformations, 6-8
DTL, 1-1, 6-4
DTSL, 6-5, 54
DU (default unknown), 6-69
Damage, component, 2-4
Digital Test Generator (DTG), 2-1, 6-1
Digital Test Library (DTL), 6-4
Digital Test Structr Lang (DTSL), 6-5
Digital test librarian (DTL), 3-4, 6-1
Diode test, 8-19
Disable procedures, 6-1

--E--

ELSE, 6-69
ELSE IF, 6-69
END, 6-69
ENTRY, 5-4
EXTRACT, 4-2
EXTernal, 7-2
Edit-on-errors, 4-8
Error limits, 8-4

--F--

FASTATG, 8-11
FInish, 6-70
FORM test. 5-12
FORMAT, 5-8, 22, 41, 8-7
FREE, 6-70
FREQ, 6-70, 8-15, 16
FVONLY, 8-21
Flagspecs, 8-4, 10

--G--

GND, 6-71
GUARD, 5-15, 28
Guarded tests, 8-1

--H--

HEADer, 6-71
HFORCE, 6-75
HOLD, 6-76

--I--

I8085, 6-31
IC test procedures, 6-12, 8-18
IDD In-Circuit, 2-4
IDR, 8-21
IF, 5-29

IF-THEN-ELSE, 6-56
IFV, 8-21
IG (input group), 6-77
IGT, 8-24
ILIM, 8-11
INDEPendent, 6-77
INHibit, 6-14, 77
INSERT, 4-2, 6-16, 57, 78
Inductance test, 8-16
Inductors, 7-1
Inhibit procedures, 6-1
Initialization section, 6-21
Insert mode, 3-5, 6
Integrated circuit test, 8-28
Item, 3-2

--K--

KNOWN, 6-79
Keywords DTSL, 6-6, 51

--L--

LBS original, 4-4
LERR, 8-4,16
LET, 6-80
LFORCE, 6-80
LIST mode, 3-4, 7, 4-2
Library design, 5-45
Library organization, 2-4

--M--

MAIN, 6-2, 17, 81
MOVE, 3-6, 4-3
Measurement error limits, 8-4
Merge mode, 2-1, 9
Microprocessor tests, 6-27

--N--

NAME, 4-1
NEED, 5-15, 30
NEGRAIL, 8-26
NEWTST, 5-22
NEXT, 5-31
NODEMOVE, 5-38
NOTEST, 7-2
NP (no preference), 6-83
Non-guarded measurement, 8-1

--O--

OG (output group), 6-84
OU (output unknown), 6-86
OUTLINE, 4-3, 5-1
OV (output value), 6-86
Opens test, 8-8
Operand functions, 5-25
Operational amplifier test, 8-24

--P--

PARAMETERS, 5-8
PHn (preferred high), 6-86
PIN, 4-7, 5-15, 32
PINCNT, 5-38
PLn (preferred low), 6-88
POSRAIL, 8-26
PRESET, 8-12
Paramspecs, 5-44
Pin states, 6-7, 84
Potentiometer test, 8-12
Procedure development, 6-12

--Q--

QUIT IGNORE, 5-13, 33, 8-6
QUIT, 5-13, 32, 6-88

--R--

R6502, 6-27
RAM test, 6-15
REMOVE mode, 3-6, 4-3
REPLACE mode, 3-6, 4-3
RERR, 5-16, 8-4, 10
RHIRESID, 8-11
RHIV, 8-11
RLORESID, 8-11
RLOV, 8-11
RTOL, 8-11
Rest, 6-29
Resistance test procedures, 8-9
Resistors, 7-1
Reverse leadkage test, 8-21

--S--

SAVE, 5-33
SAY, 5-19, 34, 6-89
SCR test, 8-22
SELECTION, 5-7, 18
SET, 5-15, 34
SET CLOCK, 6-15
SETINDependent, 6-90
SKELETON, 6-16
SPECial, 6-90
SUBroutine, 6-16, 92
SUMMARY, 4-4
Scratchprobing, 8-8
Search mask, 4-4
Shorts test, 8-8
Single-step, 6-26
Skeleton file, 3-1
SpeedPlus testing, 6-39
Statistics, 2-6
Synchronous tests, 6-26
System variables, 5-26, 39

--T--

TBOOL, 5-39
TEST, 4-3, 5-6, 15, 35
TG (trigger group), 6-92
TOL, 8-4
TRACE mode, 6-11
TYPE, 5-35
Test evaluation, 8-3
Test library, 8-1
Test procedure, sample, 6-95
Test program development, 8-1
Test selection, analog, 8-5
Test selection, digital, 6-3, 8-7
Transistor test, 8-17
Transistors, 7-1
Types of tests, 8-1

--U--

UG (unknown group), 6-93
UKnn (unknown device), 7-2
UNDEFined, 6-93
USING, 6-94
Update library, 2-4
User library file, 3-1
User test library, 2-4
Utility test procedures, 8-8

--V--

VA, 8-24
VBE, 8-18
VCC, 6-95
VCE, 8-18
VDIFF, 8-26
VERIFY, 4-3
VETO, 8-6
VLSI testing, 6-25

--W--

WOR, 2-3
WRILIM, 8-4

--Z--

Zener diode test, 8-21

GenRad

Production Test Division
300 Baker Avenue
Concord, Massachusetts 01742

USA

***Boston** • (Sales) Waltham, MA 02254
Tel: (617) 890-4900 • TWX: 710 324 0893

(Service) Concord, MA 01742
Tel: (617) 369-4400 • TWX: 710 347 1051

***Chicago** • 1156 Shure Drive, Arlington Heights, IL 60004
Tel: (312) 577-1881 • TWX: 910 291 1209

***Cincinnati** • Suite A, 2002 Ford Circle, Milford, OH 45150
Tel: (513) 831-9210 • TWX: 810 460 8323

***Colorado Springs** • 6180 Lehman Drive, Suite B207, Colorado Springs, CO 80907
Tel: (303) 593-8238

***Dallas** • 1121 Rockingham Lane, Suite 100, Richardson, TX 75080
Tel: (214) 234-3357 • TWX: 910 867 4771

***Los Angeles** • P.O. Box 19500, 17361 Armstrong Avenue,
Irvine Industrial Complex, Irvine, CA 92714
Tel: (714) 540-9830 • TWX: 910 595 1762

***New York** • 22-08 Route 208, Fair Lawn, NJ 07410
Tel: (NJ) (201) 797-8001; (NY) (212) 964-2722 • TWX: 710 988 2205

***Orlando** • 3751 Maguire Blvd., Suite 170, Orlando, Florida 32803
Tel: (305) 894-4303 • TWX: 810 850 0270

***San Francisco** • 2855 Bowers Avenue, Santa Clara, CA 95051
Tel: (408) 727-4400 • TWX: 910 338 0291

***Washington, D.C.** • 1701 Research Boulevard, Rockville, MD 20850
Tel: (301) 424-6224 • TWX: 710 828 9783

CANADA

***Toronto** • 307 Evans Avenue, Toronto, Ontario M8Z 1K2
Tel: (416) 252-3395 • TLX: 06-967624

Montreal • Tel: (514) 747-1052 • TLX: 05-826652

EUROPE

***England** • Norreys Drive, Maidenhead, Berkshire SL6 4BP England
Tel: (0628) 39181 • TLX: 851-848321

Best • Tel: (0) 49984240 • TLX: 844-59535

***Milan** • Tel: (02) 8466541 • TLX: 843-320373

***Munich** • Tel: (089) 41690 • TLX: 841-529917

***Paris** • Tel: (01) 7970739 • TLX: 842-220991

Rennes • Tel: (099) 535154

***Rome** • Tel: (06) 455839

Wiesbaden • Tel: (06121) 370057 • TLX: 841-04186489

***Zurich** • Tel: (01) 552420 • TLX: 845-53638

ASIA

Japan • Tokyo Electron Limited, Shinjuku-Nomura Bldg.,
1-26-2 Nishi-Shinjuku, Shinjuku-Ku, Tokyo 160, Japan
Tel: 343-4411 • TLX: 232-2240 Labtel J

Other countries in Asia and Pacific • Tel: 344-9245 (Tokyo)

LATIN AMERICA

Contact WALTHAM, MA 02254 USA
Cable GENRAD Waltham (MASS.)

*Service Facilities Available

Printed in USA